

T.C.  
EGE ÜNİVERSİTESİ  
Fen Bilimleri Enstitüsü

**KOPYALA TAŞI SAHTECİLİĞİ TESPİTİNE  
YÖNELİK GELİŞTİRİLMİŞ YEREL İKİLİ ÖRÜNTÜ  
ALGORİTMASI TASARIMI**

Fatih DAĞDEVİREN

Danışman : Prof. Dr. Serdar KORUKOĞLU  
İkinci Danışman : Doç. Dr. Muhammed Gökhan  
CİNSDİKİCİ

Uluslararası Bilgisayar  
Bilgi Teknolojileri Doktora Programı

İzmir  
2021



**EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ****ETİK KURALLARA UYGUNLUK BEYANI**

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum “Kopyala Taşı Sahteciliği Tespitine Yönelik Geliştirilmiş Yerel İkili Örüntü Algoritması Tasarımı” başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

... / ... / 2021

İmzası

Fatih DAĞDEVİREN



## ÖZET

### KOPYALA TAŞI SAHTECİLİĞİ TESPİTİNE YÖNELİK GELİŞTİRİLMİŞ YEREL İKİLİ ÖRÜNTÜ ALGORİTMASI TASARIMI

DAĞDEVİREN, Fatih

Doktora Tezi, Uluslararası Bilgisayar Anabilim Dalı

Tez Danışmanı: Prof. Dr. Serdar KORUKOĞLU

Haziran 2021, 116 sayfa

Teknolojinin gelişmesi ile birlikte dijital yazılımların ve yazılım araçlarının kullanımı gün geçtikçe yaygınlaşmaktadır. Yazılımlar ve yazılım araçlarındaki bu artış ve çeşitlilik görüntüler üzerinde birtakım değişiklikler yapabilmeyi mümkün kılmaktadır. Görüntü üzerindeki bir nesnenin aynı görüntüde başka bir bölgeye çeşitli yöntemler uygulanarak kopyalanmasıyla yapılan görüntü değiştirme işlemi “Kopyala Taşı Sahteciliği” olarak adlandırılmaktadır.

Çalışma kapsamında, LBP (Local Binary Pattern) ve VBSD (Variable Bin Size Distance) ile kopya hareket tespiti için yeni bir hibrit yöntem önerilmektedir. Önerilen yöntemde resimlere ait farklı bileşenler farklı LBP operatörlerinden geçirilmektedir. Elde edilen sonuçlar, çalışmanın diğer çalışmalardan farkını ortaya koyan uç uca vektör birleştirme yönteminden geçirilmesi sonucu öznitelikleri oluşturulmakta ve daha sonra eşlenmektedir. Çalışma kapsamında ilk olarak, SIFT köşe algılama algoritması ile köşe noktaları tespit edilmektedir. Daha sonra her köşe noktasının etrafında alanlar oluşturulmaktadır. Ardından, öznitelik vektörü adı verilen bir vektör oluşturmak için alanlara LBP uygulanmaktadır. Son olarak, en yakın mesafe değerine göre VBSD kullanılarak özellikler eşleştirilmektedir. Deneysel sonuçlar, hibrit yöntemin, kopyalanmış bölgeleri orjinal LBP algoritmalarından ve DAISY (A Fast Local Descriptor for Dense Matching), SURF (Speeded Up Robust Features) ve SIFT (Scale Invariant Feature Transform) gibi diğerlerinden daha doğru bir şekilde tespit edebildiğini göstermektedir.

**Anahtar sözcükler:** Kopyala Taşı Sahtecilik Tespiti, Görüntü İşleme, SIFT, LBP, VBSD, Görüntü Sahteciliği



**ABSTRACT****ENHANCED LOCAL BINARY PATTERN ALGORITHM DESIGN  
FOR COPY MOVE FORGERY DETECTION**

DAĞDEVİREN, Fatih

Phd in Int. Computer Ins.

Supervisor: Prof. Dr. Serdar KORUKOĞLU

June 2021, 116 pages

The rapid development of technology from day to day has made easier to meet with new technologies. This makes easier to manipulate original image as we want to create fake images. Thus, importance of detecting fakes images becomes evident here. Copy move forgery is one of the most widely used detection technique for image forgery.

In this study, a new hybrid method for copy move detection by LBP (Local Binary Pattern) and VBSD (Variable Bin Size Distance) is proposed. In the proposed method, different components of the images are passed through different LBP operators. As a result of the end-to-end vector joining method, which distinguishes the study from other studies, features are created and matched. First of all, within the scope of the study, corner points were discovered by SIFT corner detection algorithm. Then square like patches were created around each corner point. After, three LBP were applied on patches to create a vector called feature vector. At last, features were matched by using variable bin size distance according to closest distance value. Experimental results show that the hybrid method can detect duplicated regions with better accuracy than original local binary pattern algorithms and others like DAISY (A Fast Local Descriptor for Dense Matching), SURF (Speeded Up Robust Features) and SIFT (Scale Invariant Feature Transform).

**Keywords:** Copy Move Forgery Detection, Image Processing, SIFT, Local Binary Pattern, Variable Bin Side Distance, Image Forgery



## ÖNSÖZ

Bu çalışmada görüntülerde yapılan kopyala taşı sahtecilik işlemlerinin tespiti yapılmaktadır. Görüntü üzerindeki bir nesnenin aynı görüntüde başka bir bölgeye kopyalanmasıyla yapılan görüntü değiştirme işlemine kopyala taşı sahteciliği adı verilmektedir. Teknolojik cihazların hızla gelişmesi ve sosyal medyada paylaşılan fotoğrafların çok sayıda insana ulaşması sebebiyle artan sahteciliklerin tespit edilmesi için bu tez çalışması hazırlanmıştır.

Tez çalışmamda, planlanmasında, araştırılmasında ve yürütülmesinde ilgi ve desteğini esirgemeyen,engin bilgi ve tecrübelerinden yararlandığım sayın hocalarım Prof. Dr. Serdar KORUKOĞLU ve Doç. Dr. Muhammed Gökhan CİNSDİKİCİ'ye teşekkürlerimi sunarım.

İZMİR

.../.../2021

Fatih DAĞDEVİREN



**İÇİNDEKİLER**Sayfa

|                                                       |      |
|-------------------------------------------------------|------|
| İÇ KAPAK.....                                         | ii   |
| KABUL ONAY SAYFASI.....                               | iii  |
| ETİK KURALLARA UYGUNLUK BEYANI .....                  | iv   |
| ÖZET.....                                             | v    |
| ABSTRACT.....                                         | vi   |
| ÖNSÖZ .....                                           | vii  |
| İÇİNDEKİLER DİZİNİ .....                              | viii |
| ŞEKİLLER DİZİNİ.....                                  | x    |
| TABLolar DİZİNİ .....                                 | xv   |
| SİMGELER VE KISALTMALAR DİZİNİ .....                  | xvi  |
| 1. GİRİŞ .....                                        | 1    |
| 2. LİTERATÜR ÖZETİ.....                               | 5    |
| 3. TEMEL BİLGİLER .....                               | 11   |
| 3.1 SİFT KÖŞE TESPİT VE ÖZNİTELİK TANIMLAYICISI ..... | 11   |



## İÇİNDEKİLER (devam)

### Sayfa

|                                          |      |
|------------------------------------------|------|
| 3.2. SURF ÖZNİTELİK TANIMLAYICISI .....  | 15   |
| 3.3 DAISY ÖZNİTELİK TANIMLAYICISI .....  | 16   |
| 3.4 LBP ÖZNİTELİK TANIMLAYICISI .....    | 18   |
| 3.5 VBSD EŞLEME YÖNTEMİ .....            | 20   |
| 3.6 BENZERLİK HESAPLAMA YÖNTEMLERİ ..... | 21   |
| 3.6.1 ÖKLİT MESAFESİ .....               | 21   |
| 3.6.2 MANHATTAN UZAKLIĞI .....           | 22   |
| 3.6.3 Kİ-KARE UZAKLIĞI .....             | 22   |
| 3.6.4 JEFFREY DİVERJANSI .....           | 22   |
| 4. YÖNTEM.....                           | 23   |
| 5. BULGULAR.....                         | 29   |
| 6. SONUÇ VE ÖNERİLER .....               | 47   |
| KAYNAKLAR DİZİNİ .....                   | 49   |
| TEŞEKKÜR.....                            | xvii |
| ÖZGEÇMİŞ .....                           | 55   |
| EKLER.....                               |      |



## ŞEKİLLER DİZİNİ

| <u>Şekil</u>                                                                                                                                                                                                       | <u>Sayfa</u> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| 1.1. Gerçek hayattan alınan sahteciliğe uğramış görüntü .....                                                                                                                                                      | 1            |
| 3.1. Ölçeksel uzay ve DOG piramidi .....                                                                                                                                                                           | 12           |
| 3.2 Ölçeksel uzay ve uç noktaları .....                                                                                                                                                                            | 13           |
| 3.3. SIFT histogramı oluşumu .....                                                                                                                                                                                 | 14           |
| 3.4 DAISY öznitelik yapısı.....                                                                                                                                                                                    | 17           |
| 3.5 LBP değeri hesaplama .....                                                                                                                                                                                     | 19           |
| 3.6 Farklı LBP operatörleri.....                                                                                                                                                                                   | 19           |
| 3.7 VBSD eşleme adımları .....                                                                                                                                                                                     | 21           |
| 4.1. Sahteciliğe uğramış görüntü .....                                                                                                                                                                             | 23           |
| 4.2 Önerilen yöntem adımları .....                                                                                                                                                                                 | 24           |
| 4.3 Öznitelik çıkarma işlemleri.....                                                                                                                                                                               | 25           |
| 4.4 COMOFOFOD-26 sonuçları a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 28           |
| 5.1 Önerilen yöntem, SIFT, SURF, DAISY öteleme atağı karşılaştırılması .....                                                                                                                                       | 41           |
| 5.2 Önerilen yöntem, SIFT, SURF, DAISY döndürme atağı karşılaştırılması..                                                                                                                                          | 41           |
| 5.3 Önerilen yöntem, SIFT, SURF, DAISY ölçekleme atağı karşılaştırılması..                                                                                                                                         | 41           |



## ŞEKİLLER DİZİNİ (devam)

Şekil

Sayfa

|     |                                                                                                                                                                                                    |    |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 5.4 | Önerilen yöntem, SIFT, SURF, DAISY çarpıtma atağı karşılaştırılması .....                                                                                                                          | 42 |
| 5.5 | Önerilen yöntem, SIFT, SURF, DAISY karma atağı karşılaştırılması .....                                                                                                                             | 42 |
| 5.6 | COMOFOD-13 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 46 |
| 5.7 | COMOFOD-70 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 46 |
| A.1 | COMOFOD-1 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 62 |
| A.2 | COMOFOD-2 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 63 |
| A.3 | COMOFOD-4 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 64 |
| A.4 | COMOFOD-7 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 65 |



## ŞEKİLLER DİZİNİ (devam)

| <u>Şekil</u>                                                                                                                                                                                            | <u>Sayfa</u> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| A.5 COMOFOD-11 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 66           |
| A.6 COMOFOD-12 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 67           |
| A.7 COMOFOD-13 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 68           |
| A.8 COMOFOD-15 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 69           |
| A.9 COMOFOD-16 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 70           |
| A.10 COMOFOD-22 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 71           |
| A.11 COMOFOD-23 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 72           |



## ŞEKİLLER DİZİNİ (devam)

| <u>Şekil</u>                                                                                                                                                                                             | <u>Sayfa</u> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| A.12 COMOFOOD-24 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 73           |
| A.13 COMOFOOD-26 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 74           |
| A.14 COMOFOOD-27 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 75           |
| A.15 COMOFOOD-28 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 76           |
| A.16 COMOFOOD-29 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 77           |
| A.17 COMOFOOD-31 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 78           |
| A.18 COMOFOOD-72 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 79           |



## ŞEKİLLER DİZİNİ (devam)

| <u>Şekil</u>                                                                                                                                                                                              | <u>Sayfa</u> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| A.19 COMOFOOD-79 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 80           |
| A.20 COMOFOOD-114 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 81           |
| A.21 COMOFOOD-75 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 82           |
| A.22 COMOFOOD-50 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 83           |
| A.23 COMOFOOD-51 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası .....  | 84           |
| A.24 COMOFOOD-141 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 85           |
| A.25 COMOFOOD-167 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Tez yöntemi eşleme sonrası g) RANSAC sonrası ..... | 86           |



## TABLOLAR DİZİNİ

| <u>Tablo</u>                                                                                                                | <u>Sayfa</u> |
|-----------------------------------------------------------------------------------------------------------------------------|--------------|
| 2.1 Algoritmaların karşılaştırılması .....                                                                                  | 9            |
| 5.1 COMOFOFOD veri seti içerikleri .....                                                                                    | 29           |
| 5.2 COMOFOFOD veri seti görüntü içerikleri .....                                                                            | 29           |
| 5.3 CVIP veri seti içerikleri .....                                                                                         | 33           |
| 5.4 Önerilen yöntem (Ö.Y.) 11x11 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD), Manhattan Uzaklığı (MD) test sonuçları..... | 35           |
| 5.5 Önerilen yöntem (Ö.Y.) 13x13 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD), Manhattan Uzaklığı (MD) test sonuçları..... | 36           |
| 5.6 Önerilen yöntem (Ö.Y.) 15x15 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD), Manhattan Uzaklığı (MD) test sonuçları..... | 37           |
| 5.7 Önerilen yöntem (Ö.Y.) 17x17 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD), Manhattan Uzaklığı (MD) test sonuçları..... | 38           |
| 5.8 Önerilen yöntem (Ö.Y.) ve orjinal LBP algoritmaları karşılaştırılması..                                                 | 39           |
| 5.9 Önerilen yöntem (Ö.Y.) ve SIFT, SURF, DAISY algoritmaları karşılaştırılması .....                                       | 40           |
| 5.10 CVIP üzerinde önerilen yöntem (Ö.Y.) ve SIFT, SURF, DAISY algoritmaları karşılaştırılması .....                        | 43           |



## SİMGELER VE KISALTMALAR DİZİNİ

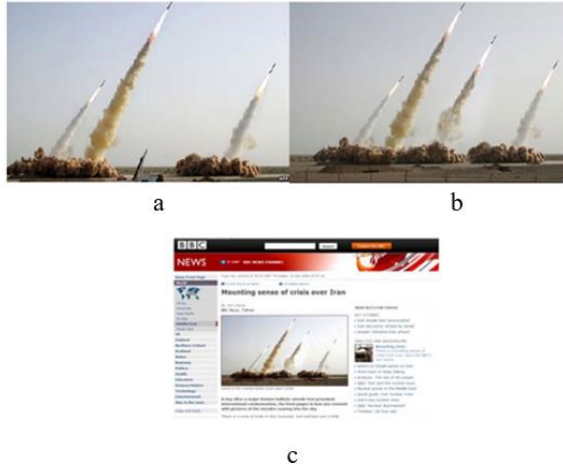
### Kısaltmalar

|         |                                                |
|---------|------------------------------------------------|
| SIFT    | Scale Invariant Feature Transform              |
| SURF    | Speeded Up Robust Features                     |
| DAISY   | A Fast Local Descriptor for Dense Matching     |
| LBP     | Local Binary Pattern                           |
| VBSD    | Variable Bin Size Distance                     |
| DCT     | Discrete Cosine Transform                      |
| PCA     | Principal Component Analysis                   |
| FMT     | Fourier-Mellin Transform                       |
| CVIP    | Computer Vision and Image Processing Group     |
| COMOFOD | Image Database for Copy-Move Forgery Detection |
| RANSAC  | Random Sample Consensus                        |



## 1.GİRİŞ

Teknolojinin günden güne gelişmesi, teknolojik cihazların insanlar tarafından temin edilmesini kolaylaştırmaktadır. Birçok alanda olduğu gibi teknolojinin bu şekilde ilerlemesi ve insanlar tarafından temin edilmesinin daha kolay hale gelmesi görüntü işleme çalışma alanını da etkilemiştir. Bu sayede görüntü işleme araçlarının kullanımı artmıştır. Bu artış iyi niyetli insanlar tarafından etik sonuçlar ortaya çıkarırken kötü niyetli insanlar tarafından kullanılmaması gereken alanlarda kullanılabilmektedir. Bu kullanım sonucunda görüntü işleme alanında sahte resimler ortaya çıkabilmektedir. Görüntüler üzerinde istemeyerek ya da kasıtlı olarak yapılan değişiklikler görüntülerin orijinal hallerini bozmaktadır. Sahte resimlerin tespiti için ise görüntü doğrulama tekniklerinden dijital kopyala taşı sahtecilik tespiti yöntemleri kullanılmaktadır. Görüntü doğrulama alanında yapılan çalışmaların amacı görüntünün bozulup bozulmadığını tespit etmektir. Şekil 1.1’de gerçek hayatta kopyala taşı sahteciliğine maruz kalmış görüntüler gösterilmektedir. Yapılan sahtecilik 2008 yılında İran füze testleri sonrasında meydana gelmiştir. Testler sonrasında basına orijinal görüntü yerine sahteciliğe uğratılmış görüntü servis edilmiştir.



Şekil 1.1 (a) Orijinal görüntü, (b) Sahteciliğe uğramış görüntü, (c) Basına yansıtılan görüntü

Görüntü sahtecilik teknikleri aktif ve pasif teknikler olmak üzere iki gruba ayrılmaktadır. Aktif yöntemlerde görüntü içerisine yerleştirilen damganın elde edilmesi durumunda görüntünün doğrulaması gerçekleştirilmektedir. Pasif yöntemlerde ise bir ön bilgiye ihtiyaç yoktur. Aktif teknikler dijital filigran ve

imza olmak üzere iki gruptan oluşmaktadır. (Kaur et al., 2016), dijital filigranın çift yönlü bir işlem olduğunu belirtmiştir. İlk olarak, filigran, gönderici tarafından oluşturulup görüntü içerisine yerleştirilmektedir. Daha sonra görüntü, alıcı tarafından taranıp sahtecilik tespiti yapılmaktadır. Dijital imzalama tekniğinde görüntü elde edilirken görüntüye ait benzersiz tanımlayıcı çıkarılmaktadır ve bu benzersiz tanımlayıcı doğrulama esnasında kullanılmaktadır (Kaur et al., 2016).

Boz vd. (2016), çalışmalarında pasif teknikleri arkalarında herhangi bir ipucu bırakmadıkları için kör teknikler olarak tanımlamaktadır. Kaur et al. (2016), çalışmalarında pasif teknikler, görüntü dilimleme ve kopyala taşı sahteciliği olarak iki gruba ayrılmıştır. Görüntü dilimleme sahtecilik yönteminde hedef görüntü üzerine kaynak görüntüden gelen herhangi bir obje veya alan kopyalanmaktadır. Bu yöntemde hedef görüntü ve kaynak görüntü birbirleri ile aynı olmak zorunda değildir (Teerakanok et al., 2019). Kopyala taşı sahteciliğinde, görüntü dilimleme sahteciliğinin aksine hedef ve kaynak görüntü aynı görüntüdür. Hedef ve kaynak görüntünün aynı görüntü olması, sahteciliğe uğrayan alan ve kopyalanan alanın aynı görüntü içinde olması manasına gelmesi demektir. Kopyalanan alanların aynı resim içinde olması sayesinde sahteciliğe uğrayan alanların benzer karakteristikler gösterebilmektedir. Kopyala taşı sahteciliğinde, bazı durumlarda kopyalanan alana görüntü üzerinde kopyalanmadan önce döndürme, ölçekleme, gürültü ekleme vb. gibi farklı son işlemler uygulanabilmektedir. Son işlemlerin sonunda sahteciliğe uğramış görüntü ortaya çıkmaktadır (Zhao et al., 2013). Son işlem uygulanarak sahteciliğe uğramış görüntülerin tespiti, kopyala taşı sahteciliği üzerine hazırlanan çalışmaların temel hedefini oluşturmaktadır.

Kopyala taşı sahteciliği için hazırlanan bu tez çalışması kapsamında, SIFT (Scale Invariant Feature Transform) köşe nokta tespit algoritması tabanlı ve geliştirilmiş LBP (Local Binary Pattern) algoritması ile VBSD (Variable Bin Size Distance) algoritmasının birleşimi olan yeni bir algoritma hazırlanmıştır. Önerilen yöntemde ilk olarak SIFT algoritması ile köşe noktalarının tespiti yapılmaktadır. Daha sonra geliştirilmiş LBP algoritması ile tüm köşe noktalarına ait öznelilikler çıkarılmaktadır. Çıkarılan öznelilikler VBSD algoritması ile eşlenmektedir. Son adımda ise RANSAC (Random Sample Consensus) algoritması ile yanlış eşleşmeler elenmekte ve kalan eşler kopyala taşı sahteciliğine uğramış eşler olarak belirlenmektedir.

Çalışmanın ikinci bölümünde literatürde daha önce kopyala taşı sahteciliği için hazırlanan çalışmalar kronolojik olarak sıralanmaktadır. Ayrıca bu bölümde bu çalışmalarda kullanılan yöntemler ve elde edilen sonuçlara yer verilmektedir. Üçüncü bölümde önerilen yöntemde kullanılan yöntemler detaylı olarak anlatılmaktadır.

Dördüncü bölümde önerilen yöntemde sunulan yöntem ayrıntılı olarak ele alınmaktadır. İlk olarak köşe noktalarının tespitinde kullanılan yöntem anlatılmaktadır. Daha sonra köşe noktalarının karakteristik yapılarını ortaya çıkaran özniteliklerin çıkarılması ve eşlenmesi ele alınmaktadır. Üçüncü ve son adımda ise yanlış eşleşen noktaların elenmesi için kullanılan RANSAC algoritması anlatılmaktadır.

Tez çalışmasının son bölümü olan sonuç bölümünde ise önerilen yöntemden elde edilen sonuçlar karşılaştırmalı olarak görselleştirilmektedir. Son bölümde ayrıca önerilen yöntemin geliştirilebilmesi için öneriler sunulmaktadır.



## 2.LİTERATÜR ÖZETİ

Son yıllarda birçok kopyala taşı sahtecilik tespiti yöntemi önerilmiştir. Bu yöntemler temel olarak blok tabanlı ve köşe noktası tabanlı olmak üzere iki gruba ayrılmaktadır. Blok tabanlı yöntemlerde görüntü üzerinde çakışan bloklar oluşturulmaktadır ve eşleme işlemi oluşturulan bloklar üzerinden gerçekleştirilmektedir. Görüntüde oluşturulan blok sayılarının fazlalığı sonucu uzun süren eşleme işlemi bu yöntemin eksik yönü olarak ön plana çıkmaktadır. Köşe noktası tabanlı yöntemlerde görüntü üzerindeki köşe noktaları tespit edilmektedir ve öznitelikler köşe noktaları etrafından kesilen bloklar üzerinden çıkarılmakta ve eşlenmektedir. Pürüzsüz yüzeylerdeki olası etkisiz tespit kabiliyeti bu yöntemin eksik yönü olarak ön plana çıkmaktadır. Hızlı olması ve etkili performansı nedeniyle bu tez çalışmasında SIFT köşe tespit algoritması tercih edilmiştir.

Kopyala taşı sahteciliği alanında ilk yapılan çalışma 2003 yılında gerçekleştirilmiştir. Fridrich et al., (2003), çalışmalarında blok tabanlı yaklaşım kullanılmıştır. Ortaya konan çalışmada görüntü 8x8'lik alt bloklara ayrılmıştır ve DCT (Discrete Cosine Transform) ile her bir bloğa ait 1x64'lük öznitelik vektörleri çıkartılmıştır. Bloklara ait öznitelikler leksikografik olarak sıralanmış ve birbirine yakın olabilecek vektörler eşlenmiştir. Önerilen yöntem döndürme ve ölçeklemeden kaynaklı değişimlere karşı etkili iken sıkıştırmadan kaynaklı değişimlere karşı etkisizdir. Bu etkisizliğin üstesinden gelebilmek için Huang et al. (2011), çalışmalarında belirli sabit değişkenler kullanıp eşleme adımı için kullanılacak olan özniteliklerin uzunluğunu kısaltmıştır. Popescu et al. (2004), çalışmalarında Fridrich'in çalışmasına benzer bir çalışma ortaya konmuştur fakat bu çalışmada DCT yerine PCA (Principal Component Analysis) kullanılmaktadır.

Bayram ve Sencar (2009) ortaya koydukları çalışmalarında blok tabanlı yaklaşımı tercih etmişlerdir. İlk olarak çıkarılan bloklar FMT'den (Fourier-Mellin Transform) geçirilmiş ve tek boyutlu öznitelik vektörü çıkartılmıştır. Deneysel sonuçlarda FMT ve blok tabanlı kopyala taşı sahtecilik tespitinin döndürme atağına karşı dayanıklı olduğu gözlemlenmiştir.

Guo Z. et al. (2010), çalışmalarında CLBP (Completed Local Binary Pattern) adı verilen yeni bir yöntem önermişlerdir. Ortaya konan bu çalışmada merkezi gri seviye farkı, bölgesel yoğunluk farkı ve orijinal LBP algoritmaları öznitelik çıkarımı için kullanılmıştır. Sonuç olarak, ortaya konan yöntemin döndürme atağına karşı dayanıklı olduğu gözlemlenmiştir. Bunun yanında, merkezi gri seviye farklarının dahil edilmesinin yöntemi gürültü atağına karşı etkisiz kıldığı belirtilmektedir.

Bravo-Solorio et al. (2011), çalışmalarında Log-Polar koordinatları tabanlı yeni bir yöntem ortaya çıkarmıştır. Bu çalışmada ilk olarak Fridrich (2003) çalışmasında olduğu gibi görüntü bloklara ayrılmıştır. Bir sonraki adımda Log-Polar koordinatları kullanılarak bloklar eşlenmiştir. Deneysel sonuçlarda blok tabanlı yaklaşımın eşleştirme sistemine çok büyük bir yük getirdiği belirtilmektedir.

Zheng N. et al. (2011), çalışmalarında blok tabanlı ve LBP algoritmasını kullanarak döndürme atağına karşı dayanıklı bir yöntem sunmuştur. Bu çalışmada farklı olarak görüntünün gri seviye bileşeni çıkarılmamıştır. Bu nedenden dolayı eşleşme sürelerinin yüksek çıkması çalışmanın zayıf yönü olarak belirtilmektedir. Aynı zamanda çalışmada yöntemin, döndürme atağının yanında gürültü ve bulanıklaştırma ataklarına karşı dayanıklı olduğu vurgulanmaktadır.

AlSawadi M. et al., 2013 yılında ortaya koydukları çalışmalarında LBP ve en yakın komşuluk algoritmalarını birleştiren yeni bir yöntem sunulmuştur. Ortaya konan çalışmada görüntünün gri seviye birleşeninin yerine görüntüye ait kırmızı, yeşil ve mavi bileşenleri öznitelik çıkarımı için kullanılmıştır. Öznitelik çıkarımı için LBP algoritması kullanılmıştır. Çalışma sonunda yöntemin ölçekleme ve döndürme ataklarına karşı dayanıklı olduğu tespit edilmiştir. Bunun yanında kırmızı, yeşil ve mavi bileşenlerinin ayrı ayrı kullanılmasının eşleme performansını yavaşlattığı çalışma sonunda ortaya çıkan problem olduğu vurgulanmıştır.

Davarzani et al. (2013), kopyala taşı sahtecilik tespiti için hazırlanan çalışmalarında çok ölçekli LBP algoritması kullanılmıştır. Çalışmada ilk olarak görüntünün gri seviye bileşeni çıkarılmış ve ardından görüntü bloklara ayrılmıştır. Daha sonra Wiener filtresi eşleştirme performansını arttırmak bloklar üzerine uygulanmıştır. Daha sonra leksikografik sıralama yapılmış ve en yakın komşuluk

algoritması ile eşleme işlemi yapılmıştır. Çalışma sonunda yanlış eşlemeler ortadan kaldırılmıştır. Yapılan deneyler sonucunda ortaya konan çalışma döndürme, ölçekleme, JPEG sıkıştırma ve bulanıklaştırma ataklarına karşı dayanıklı bulunmuştur. Ayrıca çok ölçekli yaklaşımdan dolayı eşleme işleminin uzun sürmesi çalışmanın zayıf yönü olarak belirtilmiştir.

Üstübioğlu ve arkadaşlarının 2015 yılında ortaya koydukları çalışmalarında DCT ve LBP algoritmaları birlikte kullanılmıştır. Çalışmada ilk olarak görüntü bloklara bölünmüş ve bloklara ait enerji değerleri DCT kullanılarak hesaplanmıştır. Bu çalışmada aynı zamanda Fridrich (2003) ve Davarzani et al. (2013) çalışmalarında olduğu gibi leksikografik sıralama yapılmıştır. Yapılan deneyler sonucunda hazırlanan yöntemin düşük kalitedeki görüntülerde dahi daha iyi performans verdiği belirtilmiştir.

Bo X. Et al. (2010) çalışmalarında blok tabanlı yaklaşım yerine köşe noktası tabanlı yaklaşım kullanılmıştır. Çalışmada köşe noktalarının çıkarılması ve özniteliklerin eşlenmesi için SURF (Speeded Up Robust Features) algoritması kullanılmıştır. Yapılan deneysel sonuçlarda sahtecilik tespiti işleminin küçük görüntülerde beklenen hızda olduğu tespit edilmiştir.

Shivakumar et al. (2011) tarafından Harris köşe nokta tespit algoritması ve SIFT öznitelik eşleme algoritması tabanlı yeni bir çalışma ortaya konmuştur. Çalışmada ilk olarak Harris köşe noktası tespit algoritması ile köşe noktaları tespit edilmiş ve SIFT ile öznitelikler oluşturulmuştur. Son adımda öznitelikler en yakın komşuluk algoritması ile eşlenmiştir. Yapılan deneyler sonucunda yöntemin ölçekleme ataklarına karşı dayanıklı olduğu vurgulanmıştır. Bunun yanında yöntemin döndürme ataklarına karşı zayıf olması çalışmanın ileriki aşamalarda geliştirilebileceği yönü olarak vurgulanmıştır.

Lin et al. (2011), çalışmalarında DCT ve SURF tabanlı bir yöntem sunulmuştur. Çalışmada SURF öznitelikleri DCT katsayıları ile çarpıldıktan sonra eşleme işlemi yapılmış ve deneysel sonuçlar UCID (An Uncompressed Image Dataset) veri seti üzerinde test edilmiştir. Bu çalışmaya benzer bir çalışma Zhang et al. (2012) tarafından ortaya konmuştur. Çalışmada SURF ve FMT algoritmaları öznitelikler üzerinde birlikte kullanılmıştır. İlk olarak görüntü üzerinde düz ve düz olmayan alanlar olmak üzere iki farklı grup oluşturulmuştur. Daha sonra düz olmayan alanlara SURF, düz alanlara ise FMT uygulanarak öznitelikler

tamamlanmıştır. Yapılan deneysel sonuçlarda yöntemin büyük boyutlu görüntüler üzerinde eşleme işleminin uzun olduğu sonucuna varılmıştır.

Zhao et al. (2013), çalışmalarında köşe noktalarının tespiti için Harris köşe noktası tespit algoritması kullanılmıştır. İlk olarak görüntü uyarlanabilir Wiener filtresinden geçirilmiş ve daha sonra Harris köşe noktası tespit algoritması ile köşe noktaları tespit edilmiştir. Öznitelikler LBP algoritması kullanılarak çıkarılmıştır ve eşleme işlemi için BBF (Best Bin First) algoritması kullanılmıştır. Son olarak RANSAC algoritması ile yanlış eşlemeler elenmiştir. Sonuç olarak önerilen yöntem döndürme, bulanıklaştırma ve JPEG sıkıştırma ataklarına karşı dayanıklı olduğu tespit edilmiştir.

Prasad et al. (2016), çalışmalarında SIFT, SURF ve HOG (Histogram of Oriented Gradient) yöntemlerinin kopyala taşı sahtecilik performanslarını karşılaştırmıştır. Yapılan deneyler sonucunda en iyi eşleme performansı SIFT algoritmasının verdiği gözlemlenmiştir. Bunun yanında SURF algoritması ise en hızlı eşleme performansı veren algoritma olarak bu çalışmada bahsedilmiştir.

Muzaffer ve arkadaşlarının 2018 yılında ortaya koydukları çalışmada SIFT ve SURF algoritmalarının karışımı olan hibrit bir algoritma kullanılmıştır. Çalışmada ilk olarak SURF köşe noktaları tespit edilmiş ve bu noktalar SIFT köşe noktaları ile birleştirilmiştir. Bir sonraki adımda tespit edilen noktalara ait SURF ve SIFT öznitelikleri çıkarılıp birleştirilmiştir. Daha sonra birleştirilen öznitelikler normalize edilmiş ve işlenmiştir. Son adımda RANSAC algoritması ile yanlış eşlemeler elenmiştir. Sonuç olarak önerilen yöntemin döndürme, bulanıklaştırma, gürültü ve jpeg sıkıştırma saldırılarına karşı dayanıklı olduğu belirtilmiştir.

Savran ve arkadaşlarının 2018 yılında hazırlamış oldukları yöntemde FAST köşe tespit algoritması ile köşe noktaları çıkarılmıştır. Bir sonraki adımda SIFT algoritması ile öznitelikler çıkarılmış ve Öklid uzaklığı metriği ile eşleme yapılmıştır. Son adımda yanlış eşlemeler RANSAC algoritması ile elenmiştir. Bunun yanında çalışmada farklı paralel programlama teknikleri test edilmiştir.

Kuznetsov 2019 yılında derin öğrenme temelli kopyala taşı sahteciliği çalışması hazırlamıştır. Bu çalışmada CASIA veri seti üzerindeki görüntüler derin öğrenme için %80 oranında ayrılmış ve derin öğrenme modeli ayrılan resimler üzerinde test edilmiştir. Geri kalan %20'lik dilim içerisine giren resimler test

işlemleri için kullanılmıştır. Kuznetsov öznetelik oluşturmak için CNN tabanlı derin öğrenme modelinden yararlanmıştır. Sonuç olarak önerilen yöntemin çarpıtma atağına karşı %97,8 oranında başarılı olduğu gözlemlenmiştir. Fakat çalışma içerisinde diğer ataklara karşı bir bildirim yapılmaması çalışmanın geliştirilecek yönleri arasındadır.

Agarwal et al. (2020), derin öğrenme tabanlı bir yöntem geliştirmiştir. Yöntem içerisinde ilk olarak görüntü renk değerlerine göre bloklara bölünmektedir. Daha sonra bloklar üzerinden bir derin öğrenme olan VGGNET (Visual Geometry Group-net) kullanarak öznetelikler oluşturulmuştur. Son adımda ise oluşturulan öznetelikler eşlenmektedir. Sonuç olarak çalışmanın döndürme, ölçeklendirme ve gürültü ataklarına karşı başarılı sonuçlar verdiği gözlemlenmiştir. Ayrıca birden fazla kopyalanan alanın tespiti yapılmaması çalışmanın zayıf yönü olarak belirtilmiştir.

Kopyala taşı sahtecilik tespiti alanında yapılan ve yukarıda bahsedilen çalışmalara ait özet Tablo 2.1’de gösterilmektedir.

Tablo 2.1 Algoritmaların karşılaştırması

| Çalışma              | Yöntem    | Sonuç                                                                                                                                        |
|----------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Fridrich et al.      | DCT       | Döndürme, ölçekleme ataklarına karşı dayanıklı, Sıkıştırma konusunda zayıf olduğu belirtilmiştir.                                            |
| Popescu              | PCA       | JPEG sıkıştırma, gürültü ataklarına karşı dayanıklı olduğu sonucuna varılmıştır fakat diğer teknikler için bir çalışma yapılmamış            |
| Bayram ve Sencar     | FMT       | Döndürme atağına karşı dayanıklı olduğu belirtilmektedir.                                                                                    |
| Guo Z., et al.       | CLBP      | Çalışmanın döndürme atağına karşı dayanıklı, gürültü atağına karşı zayıf olduğu belirtilmektedir.                                            |
| Bravo Solorio et al. | Log-Polar | Çalışmada blok tabanlı ve log-polar yaklaşımının eşleşme işlemini çok uzun sürede gerçekleştirdiği sonucuna varılmıştır.                     |
| Zhang N. et al.      | LBP       | LBP algoritması ile döndürme atağına karşı yeni bir yöntem tasarlanmıştır. Çalışma sonucunda ortaya konan yöntemin eşleşme işlemini çok uzun |

|                           |                   |                                                                                                                                                                                                                                                                                                |
|---------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           |                   | sürede gerçekleştirdiği sonucuna varılmıştır.                                                                                                                                                                                                                                                  |
| AlSawadi et al.           | LBP + BBF         | Hazırlanan yöntemin ölçekleme ve döndürme ataklarına karşı dayanıklı olduğu sonucuna varılmıştır. Bunun yanında görüntüye ait bileşenlerin ayrı ayrı kullanılmasının eşleşme süresini arttırdığı sonucuna varılmıştır.                                                                         |
| Davarzani et al.          | MLBP              | Hazırlanan yöntemin döndürme, ölçekleme, JPEG sıkıştırma ataklarına karşı dayanıklı olduğu vurgulanmıştır. Bunun yanında çalışmada kullanılan çok ölçekli yaklaşımın eşleşme süresini arttırdığı belirtilmektedir.                                                                             |
| Üstübioğlu ve arkadaşları | DCT + LBP         | Çalışma sonunda yöntemin büyük boyutlu görüntülerde istenen hızda olmadığı vurgulanmaktadır.                                                                                                                                                                                                   |
| Shivakumar et al.         | Harris + SIFT     | Çalışma sonunda hazırlanan yöntemin ölçekleme atağına karşı dayanıklı fakat döndürme atağına karşı zayıf olduğu ortaya konmuştur.                                                                                                                                                              |
| Lin et al.                | DCT + SURF        | Büyük boyutlu görüntülerde eşleme işleminin uzun sürmesi çalışmanın zayıf yönü olarak belirtilmektedir.                                                                                                                                                                                        |
| Muzaffer ve arkadaşları   | SIFT + SURF       | Önerilen yöntemin döndürme, bulanıklaştırma, gürültü ve JPEG sıkıştırma ataklarına karşı dayanıklı olduğu belirtilmiştir.                                                                                                                                                                      |
| Kuznetsov                 | Derin Öğrenme     | Çalışmanın çarpıtma atağındaki başarı oranı %97,8 olarak tespit edilmiştir. Fakat diğer ataklar hakkında bir bildirim yapılmamıştır.                                                                                                                                                           |
| Agarwal et al.            | Derin Öğrenme     | Çalışma döndürme, ölçekleme ve gürültü ataklarına karşı rakiplerinden iyi durumda olduğu belirtilmiştir. Aynı görüntü içerisinde birden fazla kopya tespiti yapamaması çalışmanın zayıf yönü olarak belirtilmektedir.                                                                          |
| Tez Çalışması             | SIFT + LBP + VBSD | Alanında ilk defa birlikte kullanılan LBP ve VBSD algoritmaları ile hazırlanan yöntem öteleme, ölçekleme, çarpıtma, döndürme ve karma ataklarda başarılı sonuçlar vermektedir. İlerleyen aşamalarda doğru eşleşme sayısının artması çalışmanın daha başarılı sonuçlar vermesini sağlayacaktır. |

### 3.TEMEL BİLGİLER

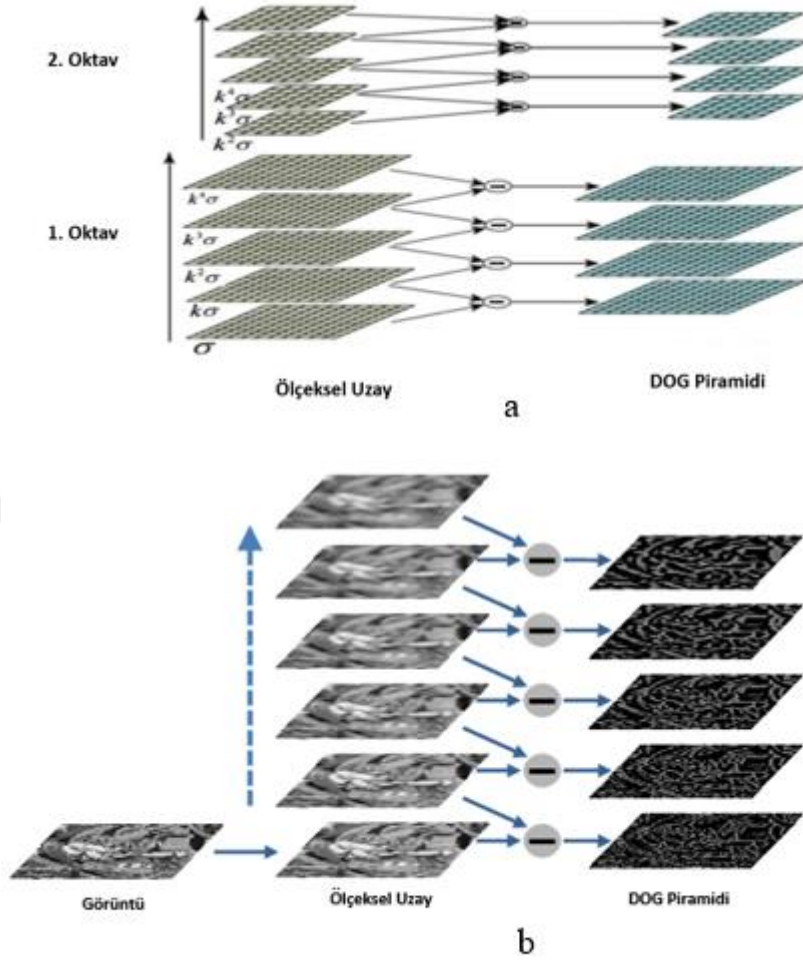
Bu bölümde literatürde adı geçen köşe noktalarının tespiti ve özniteliklerin oluşturulmasında kullanılan SIFT, SURF, DAISY, LBP algoritmaları detaylı olarak anlatılmaktadır. Ayrıca oluşturulan özniteliklerin eşlemesinde kullanılan Ki-kare testi, Manhattan uzaklığı, Öklid uzaklığı ve Jeffrey diverjansı uzaklık formülleri ve tez çalışmasında kullanılan VBSD algoritması detaylı olarak açıklanmaktadır.

#### 3.1 SIFT KÖŞE TESPİT VE ÖZNİTELİK TANIMLAYICISI

SIFT Lowe tarafından 1999 yılında sunulan bir görüntü işleme algoritmasıdır. SIFT köşe nokta tespiti ve özniteliklerin çıkarılması olacak şekilde iki parçadan oluşmaktadır. Tez kapsamında köşe noktalarının tespiti için SIFT algoritmasından yararlanılmıştır. Çünkü SIFT ölçeksel değişimlere karşı dayanıklı olduğu için diğer algoritmalara göre daha etkilidir (Ashwini, 2014).

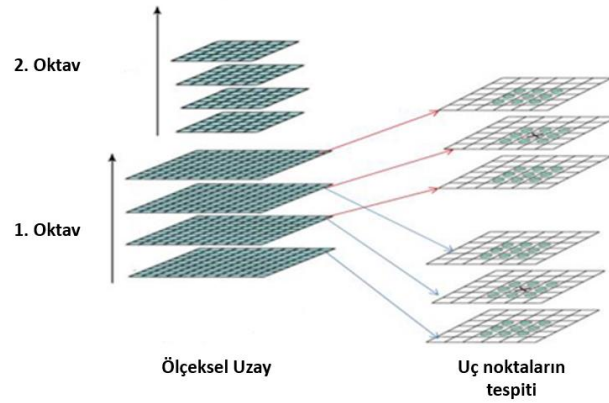
SIFT yönteminde ilk olarak köşe noktalarının tespiti yapılmaktadır. Köşe noktalarının tespitinde ilk olarak ölçeksel uzay adı verilen, aynı görüntünün farklı büyüklükteki kopyalarının farklı derecelerde Gauss bulanıklaştırmasına maruz kalmış kopyalarının tutulduğu gruplar oluşturulmaktadır. Bu gruplara Oktav adı verilmektedir (Lowe, 2003). Gauss bulanıklaştırmasında kullanılan formül  $k^x\sigma$  olarak belirlenmiştir. Farklı kademeler için  $x$  değişkeni kademeli olarak artmaktadır. Lowe çalışmasında 3 oktav ve oktavların içinde 5 farklı kopya bulunmasını önermiştir. Oktav içinde bulunan kopyalar bir önceki oktavda bulunan kopyaların büyüklük olarak yarısına eşittir.

SIFT yönteminde ölçeksel uzayın oluşturulmasından sonra DOG (Difference of Gaussian) piramidi oluşturulmaktadır. DOG piramidi aynı oktavda bulunan ardışık görüntülerin değerlerinin birbirlerinden çıkarılması ile oluşan yapıya verilen isimdir. Şekil 3.1’de ölçeksel uzay ve DOG piramidine ait görsel sunulmaktadır.



Şekil 3.1 (a) Ölçeksel uzay ve DOG piramidi (Dağdeviren, 2015) (b) Ölçeksel uzay ve DOG piramidinin görüntü üzerinde oluşturulması (Wang et al., 2010)

Lowe (2003), çalışmasında DOG piramidinden sonra lokal uç noktaların tespitinin yapıldığını belirtmektedir. Lokal uç noktaların tespiti için aynı oktavda bulunan resimler kendi içlerinde ardışık üçerli seçilerek gruplandırılmaktadır. Gruplarda orta sırada yer alan görüntüler üzerinde tarama işlemi başlatılır. Tarama işleminde orta görüntü üzerinde yer alan piksel, seçilen pikselin etrafında bulunan sekiz komşu piksel ve aynı grupta olup diğer görüntülerde aynı konumda bulunan pikseller karşılaştırılmaktadır. Karşılaştırma işleminde orta sıradaki görüntü üzerinde bulunan pikselin diğer pikseller arasında maksimum ya da minimum olması durumunda seçilen piksel uç noktası olarak işaretlenmektedir (Chia et al., 2004). Şekil 3.2’de uç noktalara ait hesaplama işlemi gösterilmektedir.



Şekil 3.2 Ölçeksel uzay ve Uç noktaları (Dağdeviren, 2015)

SIFT köşe tespit algoritmasında son adım olarak düşük kontrast değerine sahip uç noktalarının elenmektedir. Düşük kontrast değerine sahip uç noktaları DOG fonksiyonu Taylor serisinin ikinci dereceden açılımı kullanılarak elenmektedir. Bir sonraki aşamada uç noktalar arasından kenar noktalarının tespiti yapılmaktadır. Kenar noktalarının tespitinde Hessian matrisi kullanılmaktadır. Hessian matrisinden geçirilen uç noktalara ait çıkan sonucun belirli bir eşik değerinden büyük olması durumunda noktalar elenmektedir. 2003 yılında Lowe tarafından yapılan çalışmalar sonucu en uygun eşik değeri “10” olarak tespit edilmiştir. Bu işlemler sonucunda geriye kalan noktalar köşe noktaları olarak belirlenmektedir.

SIFT yönteminde köşe noktalarının tespiti tamamlandıktan sonra özniteliklerin oluşturulması işlemi gerçekleştirilmektedir. SIFT öznitelik tanımlayıcısı yaygın kullanılan görüntü işleme yöntemlerindendir. SIFT, temel olarak görüntü üzerinde seçilen piksele ait oryantasyon ve büyüklük değerlerini kullanmaktadır.

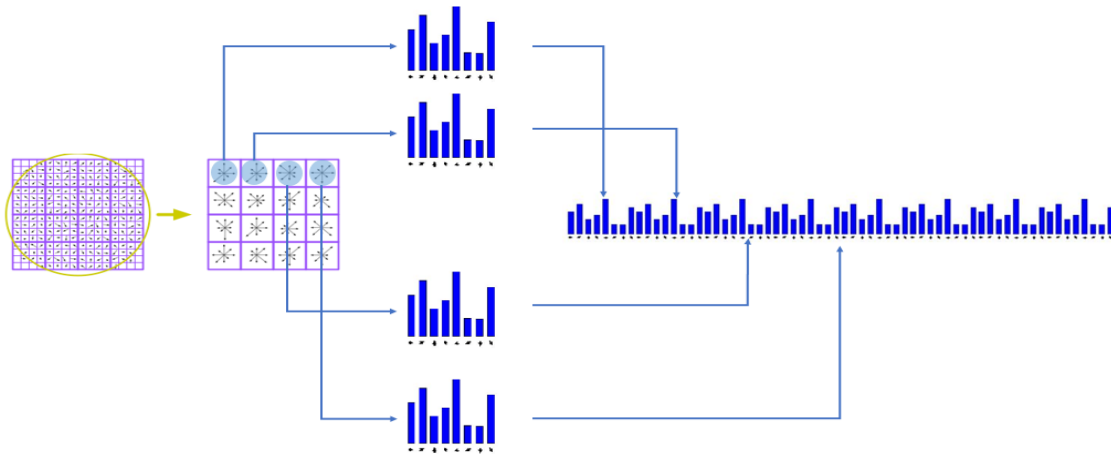
SIFT öznitelik tanımlayıcısı iki bölümden oluşmaktadır. İlk bölümde görüntü üzerinde seçilen piksel merkez kabul edilecek şekilde piksel etrafında 16x16 büyüklüğünde dörtgensel alan oluşturmaktadır. Daha sonra alanlar içerisindeki piksellere ait büyüklük ve oryantasyon değerlerini hesaplanmaktadır. Denklem 3.1 ve 3.2’de büyüklük ve oryantasyon değerlerine ait hesaplama yöntemi gösterilmektedir (Dağdeviren, 2015).

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (3.1)$$

$$\theta(x, y) = \arctan \left( \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right) \quad (3.2)$$

Denklem 3.1 ve 3.2’de SIFT öznelilik tanımlayıcısında kullanılan büyüklük ve oryantasyon değerleri gösterilmektedir. Denklem 3.1’de belirtilen  $m(x, y)$  değeri  $L$  görüntüsünde  $x$  ve  $y$  koordinatlarında bulunan piksele ait gradyan değerini,  $\theta(x, y)$  değeri  $L$  görüntüsünde  $x$  ve  $y$  koordinatlarında bulunan piksele ait oryantasyon değerini vermektedir (Hemlata, 2013).

SIFT yönteminde dörtgensel alanlar içindeki piksellere ait büyüklük ve oryantasyon değerleri hesaplanır ve daha sonra hesaplanan değerler 128’lik SIFT histogramını oluşturmak için kullanılmaktadır (Peng et al., 2011). SIFT histogramının oluşması için ilk olarak 16x16’lık dörtgensel alanlar 4x4’lük küçük alanlara bölünmektedir. Daha sonra ortaya çıkan 16 bölümün her birinden 0°-360° arasında 45°’lik farklarla oluşturulmuş 8 bölümlü histogramlar elde edilmektedir. Histogramlar oluşturulurken aynı oryantasyon değerine sahip piksellerin büyüklük değerleri toplanır ve histogramda gerekli alana yerleştirilmektedir. SIFT yönteminde son adımda her bir alan için oluşturulan histogramlar uç uca eklenmektedir. Bu yöntem sonucunda 8 bölmeli 16 histogram uç uca eklenip 128 birimlik SIFT histogramını oluşturmaktadır (Dağdeviren, 2015). Şekil 3.3’te 128 birimlik SIFT histogramının oluşum aşamaları gösterilmektedir.



Şekil 3.3 SIFT histogramı oluşumu

SIFT yöntemine ait sözde kod parçası Algoritma 1’de gösterilmektedir.

*Başla*

**Her bir köşe noktası için:**

*Adım 1: Başlangıç durumunda  $0^\circ$ - $360^\circ$  arasında  $45^\circ$  aralıklarla 8 bölüm oluştur  $\{0^\circ$ - $45^\circ$ ,  $45^\circ$ - $90^\circ$ ,  $90^\circ$ - $135^\circ$ ,  $135^\circ$ - $180^\circ$ ,  $180^\circ$ - $225^\circ$ ,  $225^\circ$ - $270^\circ$ ,  $270^\circ$ - $315^\circ$ ,  $315^\circ$ - $360^\circ$  }.*

*Adım 2: Tüm ilgi noktaları için boş SIFT histogramları oluştur.*

*Adım 3: Görüntüdeki Her bir İlgi noktası ( $P_i$ ) için:*

*Adım 4:  $P_i$  etrafında dörtgensel alan ( $W_i$ ) oluştur.*

*Adım 5  $W_i$  içerisindeki her bir piksel değeri için:*

*Adım 6: Piksel değerine ait oryantasyon değerini ( $X_i$ ) bul ve bu değere ait uygun sepet değerini oluştur.*

*Adım 7:  $X_i$ ’ye ait büyüklük değerini hesapla*

*Adım 8: 6,7’te hesaplanan değerleri  $W_i$ ’ye ait SIFT histogramına yerleştir.*

*Adım 9:  $W_i$ ’ye ait 128 birimlik SIFT histogramını oluştur.*

*Bitir*

Algoritma 3.1: SIFT tanımlayıcısı sözde kodu

### 3.2 SURF Öznitelik Tanımlayıcısı

SURF tanımlayıcısı 2008 yılında Herbert Bay ve arkadaşları tarafından sunulan bir benzerlik tespit yöntemidir. SURF yönteminde öznitelik oluşturmak için 2 aşama bulunmaktadır (Xuan Lu et al., 2010).

SURF tanımlayıcısının ilk aşamasında ilgi noktası etrafında dairesel bir alan oluşturulmaktadır. Oluşturulan dairesel alanlar içerisindeki piksel değerleri x ve y eksen yönlerinde dalgacık (*İng.* Wavelet) dönüşümünden geçirilip, Gauss fonksiyonuna tabi tutulmaktadır. Elde edilen yapı kullanılarak dairesel alan üzerinde ( $\pi/3$ ) derecelik kayan pencere kullanılarak baskın olan açı hesaplanmaktadır (Xu Bo, 2010).

SURF tanımlayıcısının ikinci adımında köşe noktası etrafında dörtgensel alanlar oluşturulmaktadır. Oluşturulan bu alanlar kendi içerisinde  $4 \times 4$ ’lük alt alanlara bölünmektedir. Oluşan her alt alana ait x ve y eksenleri yönünde, Gauss ve dalgacık dönüşümünden geçirilmektedir.

Bu işlemler sonucunda bir alt alan için 4 birim uzunluğunda bir vektör oluşmaktadır. Toplamda 16 alt alan oluştuğu için ilgi noktasına ait öznitelik 64 birim uzunluğunda bir vektör haline gelmektedir. Vektöre ait formül 3.3'te belirtilmektedir. SURF tanımlayıcısına ait sözde kod parçası Algoritma 3.2'de gösterilmektedir

*Başla*

***Her bir köşe noktası için:***

*Adım 1: Köşe noktası etrafında dairesel alan oluştur (Di).*

*Adım 2: Di üzerinde x ve y eksen yönlerinde dalgacık dönüşüm değerini hesapla.*

*Adım 3: 1'de elde edilen yapıyı gauss fonksiyonundan geçir.*

*Adım 4: Di üzerinde, ( $\pi/3$ ) derecesinde kayan pencere kullanarak baskın oryantasyon değerini bul (Oi).*

*Adım 5: İlgi noktası etrafında dörtgensel alan oluştur (Di).*

*Adım 6: Dörtgensel alanı Oi kadar rotate et.*

*Adım 7: Di'I 5x5'lük alt alanlara böl*

*Adım 8: 7'de oluşturulan her bir alt alanı saç dalgacık dönüşümünden geçir.*

*Adım 9: Dönüşümden geçen alanlara gauss uygula.*

*Adım 10: Gauss uygulanmış alanlar kullanılarak dört vektörlü yapıyı oluştur.*

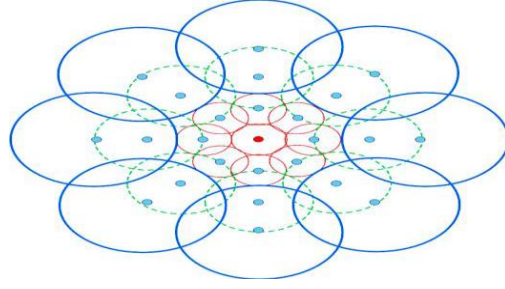
*Bitir*

Algoritma 3.2: SURF tanımlayıcısı sözde kodu

### 3.3 DAISY Öznitelik Tanımlayıcısı

DAISY tanımlayıcısı, görüntü üzerindeki özniteliklerin çıkarılmasında kullanılan 2010 yılında Engin Tola ve arkadaşları tarafından sunulan tanımlayıcıdır. DAISY temel olarak SIFT tanımlayıcısı gibi büyüklük değerleri kullanmaktadır (Sci-Kit, 2016).

DAISY tanımlayıcısında ilk olarak özniteliği oluşturulacak bölgeye ait oryantasyon haritaları çıkarılmaktadır. Oryantasyon haritalarını tanımlayan görsel Şekil 3.4'te gösterilmektedir.



Şekil 3.4 DAISY öznelik yapısı (Li li, 2014)

Şekil 3.4'te belirtilen DAISY tanımlayıcısı yapısında kırmızı nokta köşe noktasını temsil etmektedir. DAISY tanımlayıcısında kırmızı nokta merkez kabul edilmektedir ve bu nokta etrafında üç katman oluşturmaktadır. Her katman içerisinde 8 farklı bölge bulunmaktadır. Mavi renk değerine sahip noktalar bölgelere ait merkez noktalarıdır. Merkez noktaları arasındaki açı sabit olup 45 derecedir (Li li, 2014).

Şekil 3.4'de daire içeresine alınan her bir bölge birer oryantasyon haritasını temsil etmektedir. DAISY tanımlayıcısında oryantasyon haritaları Gauss fonksiyonundan geçirilmektedir. Gauss fonksiyonunda kullanılan büyüklük parametresi aynı katmanda bulunan bölgeler için aynı kalmaktadır. Gauss büyüklük değeri katmansal yapıda içeriden dışarıya doğru artmaktadır.

DAISY tanımlayıcısında oryantasyon haritalarının oluşturulmasından sonra haritalara ait oryantasyon histogramları hesaplanmaktadır. Histogram hesaplamak için kullanılan yöntem, 3.1 ve 3.2'de belirtilen SIFT yönteminde kullanılan yöntem ile aynı yöntemdir.

Son adımda bölgeler için hesaplanan histogramlar birleştirilerek tek bir histogram haline getirilmektedir. Elde edilen yapı köşe noktasına ait öznelik olarak kabul edilmektedir. Li li, çalışmasında DAISY'nin kullandığı dairesel yapının, SIFT ve SURF'teki dikdörtgensel yapıya göre daha efektif olduğunu belirtmektedir. Li li, ayrıca DAISY tanımlayıcısının eş-ortak merkezli yapısı sayesinde görüntüdeki döndürme değişimlerine karşı dayanıklı olduğunu belirtmektedir.

DAISY tanımlayıcısına ait sözde kod parçası Algoritma 3.3'de gösterilmektedir.

*Başla*

***Her bir köşe noktası için:***

*Adım 1: 8 oryantasyon haritaları katmanları içeriden dışarıya doğru oluştur.*

***Her bir oryantasyon haritası için:***

*Adım 2: Oryantasyon haritasını Gauss fonksiyonundan geçir.*

*Adım 3: Oryantasyon ve büyüklük değerlerini hesapla*

*Adım 4: 3'te hesaplanan değerleri histograma ekle.*

*Bitir*

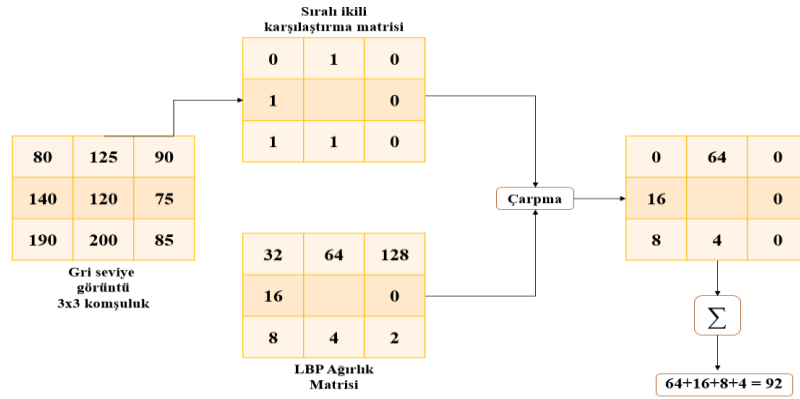
Algoritma 3.3: DAISY tanımlayıcısı sözde kodu

### 3.4 LBP Öznitelik Tanımlayıcısı

Ojala et al., 2002 yılında LBP algoritmasını ortaya çıkarmışlardır. LBP doku tarifi üzerindeki başarısı ve düşük karmaşıklık değerine sahip olmasından dolayı yaygın şekilde kullanılmaktadır. LBP tanımlayıcısında oluşan öznitelik seçilen piksel ve etrafındaki komşuları ile sıralı ikili karşılaştırılması ile ortaya çıkan değerleri içermektedir. Karşılaştırma sonucu oluşan ikili LBP matrisi, LBP ağırlık matrisi ile çarpılır ve çıkan değerlerin toplamı seçilen piksele ait LBP değeri olarak tutulmaktadır (Zhao et al., 2013). (P, R) komşuluğunda oluşan  $LBP_{P,R}$  hesaplaması denklem 3.3'te gösterilmektedir.

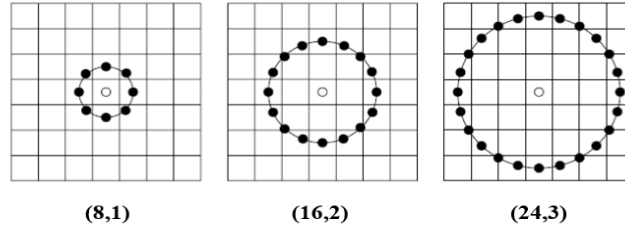
$$LBP_{P,R} = \sum_{i=0}^{p-1} s(p_i - p_c) \quad (3.3)$$

3.3'te belirtilen *LBP* komşuluk değerinde *P* değeri komşu sayısını, *R* seçilecek komşuların uzaklıkları,  $p_c$  seçilen pikseli,  $p_i$  komşu piksel değerini ifade etmektedir (Wu et al., 2017). Denklemden bulunan *s* fonksiyonundan çıkan değer komşu ve seçilen piksel değerinin farklarının 0'dan büyük eşit olma durumunda 1, diğer durumlarda 0 sonucunu vermektedir. Bir piksele ait *LBP* değerinin hesaplanması ait görsel Şekil 3.5'te gösterilmektedir.



Şekil 3.5: LBP değeri hesaplama

Ojala et al., 2002 yılında ortaya koyduğu çalışmasında farklı LBP operatörleri de hazırlamıştır. Bu operatörler seçilen piksel değeri etrafındaki uzaklığa göre farklı sayıda komşu ile sıralı ikili karşılaştırmaları yapabilmektedir. Önerilen yöntem içerisinde kullanılan LBP operatörleri Ojala'nın çalışmasında önerdiği (8,1), (16,2) ve (24,3) operatörleridir. Operatörlerdeki ilk değer komşu sayısını, ikinci değer ise komşunun seçilen piksele uzaklığını belirtmektedir. Şekil 3.6'te farklı LBP operatörlerinin görselleri gösterilmektedir.



Şekil 3.6: Farklı LBP operatörleri (Huang et al., 2011)

LBP tanımlayıcısında son adımda hesaplanan LBP değerlerine ait histograma oluşturulmaktadır. Oluşturulan histogram seçilen piksel değerini tekil kılan öznitelik olarak belirlenmektedir (Zhao et al., 2013).

LBP tanımlayıcısına ait sözde kod parçası Algoritma 3.4'de gösterilmektedir.

*Başla*

***Her bir köşe noktası için:***

*Adım 1: Köşe noktaları etrafında belirlenen büyüklükte dörtgensel alanı oluştur.*

***Dörtgensel alan içindeki her bir piksel için:***

*Adım 2: Belirlenen operatöre göre piksel etrafındaki komşuluk matrisini oluştur.*

***Her bir komşu için:***

*Adım 3: Piksel ve komşu gri seviye değerlerini karşılaştıır ve Denklem 3.4'te belirtilen "s" fonksiyonundan geçir.*

*Adım 4: 3'te çıkan değeri sıralı ikili matrisine ekle.*

*Adım 4: Sıralı ikili matrisi, LBP ağırlık matrisi ile çarp.*

*Adım 5: 4'te oluşan matris içerisindeki değerleri topla ve yeni bir matriste piksele ait yerde sakla.*

*Adım 6: LBP değerlerinin tutulduğu yeni matrise ait LBP Histogramını hesapla*

*Bitir*

Algoritma 3.4: LBP tanımlayıcısı sözde kodu

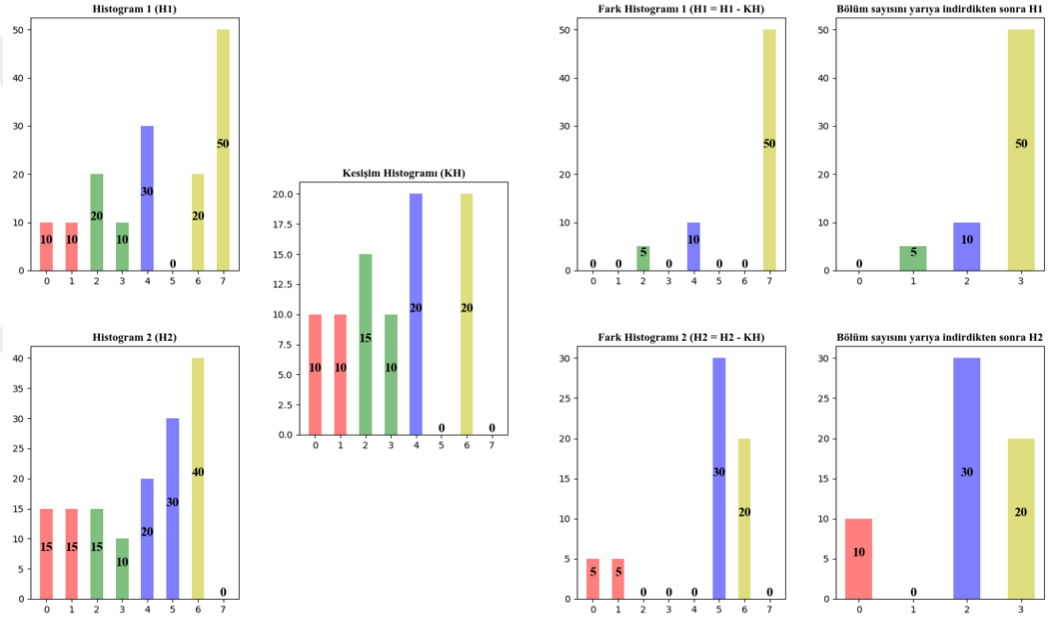
### 3.5 VBSD Eşleme Yöntemi

VBSD, Yu et al. tarafından 2010 yılında ortaya çıkan bir histograma eşleme yöntemidir. VBSD yönteminde ilk olarak histogramlar arasındaki uzaklık hesaplanmaktadır. Yu et al., çalışmasında uzaklık hesabında Ki-kare testi, Manhattan uzaklığı, Öklid uzaklığı ve Jeffrey diverjansının kullanılabileceğini belirtmiştir. Uzaklık hesabı aynı bölümlere ait değerler arasındaki fark değerlerine bakılarak hesaplanmaktadır. Bir sonraki adımda iki histograma arasındaki kesişim histogramı hesaplanmaktadır. Kesişim histogramı hesaplandıktan sonra ana histogramlardan kesişim histogramı çıkarılır ve bu sayede iki yeni fark histogramları oluşmaktadır. Fark histogramlarının hesaplanmasından sonra fark histogramları içindeki bölüm sayıları yarıya düşürülmektedir. Bir sonraki adımda fark histogramları arasındaki uzaklık hesaplanır ve daha sonra yeni kesişim histogramı hesaplanır. Yeni kesişim histogramının hesaplanmasından sonra yeni fark histogramları hesaplanmaktadır. Bu işlem "n" kadar tekrar etmektedir. Sonuç olarak "n" adet uzaklık değeri hesaplanmaktadır. Bu değerlerin ortalaması iki histograma arasındaki yakınlık bilgisini vermektedir. Denklem 3.4'te VBSD eşleme işlemine ait formül gösterilmektedir.

$$VD_x = \sum_{i=0}^{t_{\max}} (w_t \cdot d_t) \quad (3.4)$$

Denklem 3.4’te belirtilen  $VD_x$ ,  $t_{\max}$  kadar tekrarlanan işlemlerin toplamını belirtmektedir. Denklemde bulunan  $w_t$  ise histogram bölümlerinin ağırlığını ifade etmektedir.

Yu et al., sonuç olarak çalışmasında VBSD histograma eşleme yönteminin görüntüler üzerinde ışıktan (histograma öteleme), döndürme ve gürültüden kaynaklı değişimlere karşı dayanıklı olduğunu belirtmektedir. Şekil 3.7’de VBSD histogram işlemlerine ait adımlar detaylı şekilde gösterilmektedir.



Şekil 3.7: VBSD eşleme adımları

### 3.6 Benzerlik Hesaplama Yöntemleri

#### 3.6.1 Öklit Mesafesi

İki nokta arasındaki mesafeyi belirten ölçü birimidir. Basitçe iki noktaya ait  $x$  ve  $y$  koordinatlarının farklarının hipotenüsünün alınmasıyla bulunur. Denklem 3.5’te Öklit mesafesine ait hipotenüs hesaplaması gösterilmektedir.

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (3.5)$$

Denklem 3.5’da ifade edilen denklemde  $x$  ve  $y$  mesafesi hesaplanacak noktalar,  $d$  ise mesafeyi temsil etmektedir.

### 3.6.2 Manhattan Uzaklığı

Öklit gibi iki nokta arasındaki uzaklık ölçme yöntemidir. Öklit mesafesine benzer fakat farkı sadece iki nokta arasındaki farkın mutlak değeri alınır. Denklem 3.6’da Manhattan uzaklığına ait hesaplama gösterilmektedir.

$$d(x, y) = \sum_{i=1}^n |x_i - y_i| \quad (3.6)$$

Denklem 3.6’da ifade edilen denklemde  $x$  ve  $y$  mesafesi hesaplanacak noktalar,  $d$  ise mesafeyi temsil etmektedir.

### 3.6.3 Ki-kare Uzaklığı

Ki-kare uzaklığı iki nokta arasındaki benzerliği ölçme yöntemidir. Görüntü işleme dahil pek çok alanda kullanılmaktadır. Denklem 3.7’de Ki-kare uzaklığına ait hesaplama gösterilmektedir.

$$d(x, y) = \frac{1}{2} \sum_{i=1}^n \frac{(x_i - y_i)^2}{(x_i + y_i)} \quad (3.7)$$

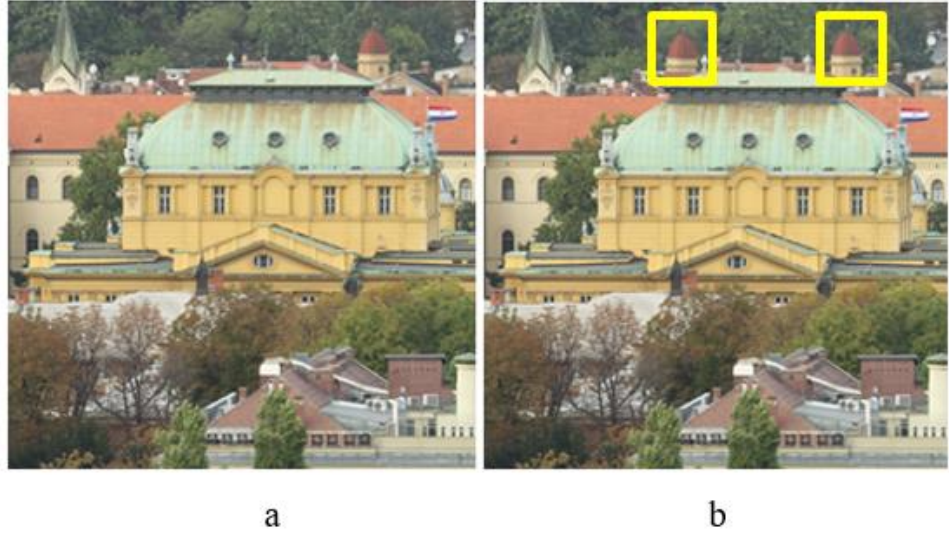
### 3.6.4 Jeffrey Diverjansı

Jeffrey Diverjansı, Kullback-Leibler diverjansının simetrik versiyonu olarak kabul edilen, radar sinyal işleme dahil pek çok alanda kullanılan bir uzaklık hesabıdır. Denklem 3.8’de Jeffrey diverjansına ait hesaplama gösterilmektedir.

$$d(x, y) = \sum_{i=1}^n \left( x_i \log \frac{x_i}{y_i} + y_i \log \frac{y_i}{x_i} \right) \quad (3.8)$$

#### 4.YÖNTEM

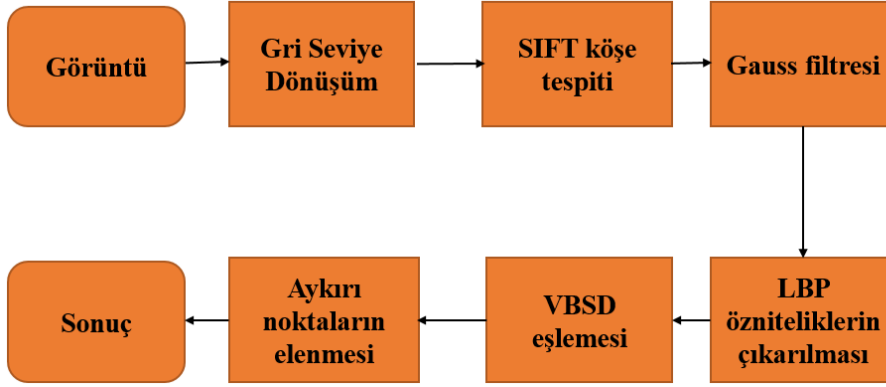
Bu bölümde ilk olarak kopyala taşı sahtecilik tespiti kapsamında hazırlanan önerilen yönteme ait adımlar detaylı şekilde anlatılmaktadır. Daha sonra eşleme işlemi için kullanılan yöntemin farklı parametreler ile test edilerek kopyala taşı sahtecilik tespitinde en iyi sonucu veren değerlere ait veriler gösterilmektedir. Bir sonraki adımda ise önerilen yöntemin orjinal LBP yöntemleri ile kıyaslaması yapılmaktadır. Son kısımda ise tez çalışmasında kullanılan yönteme ait bilgiler literatürde bulunan SIFT, SURF, DAISY, LBP (8,1), LBP (16,2), LBP (24,3), yöntemleri ile kıyaslanmaktadır.



Şekil 4.1: (a) Orijinal Görüntü (b) Bozulmuş Görüntü

Önerilen yöntemde kopyala taşı sahtecilik tespiti 4 adımda gerçekleştirilmektedir. Bu adımlar:

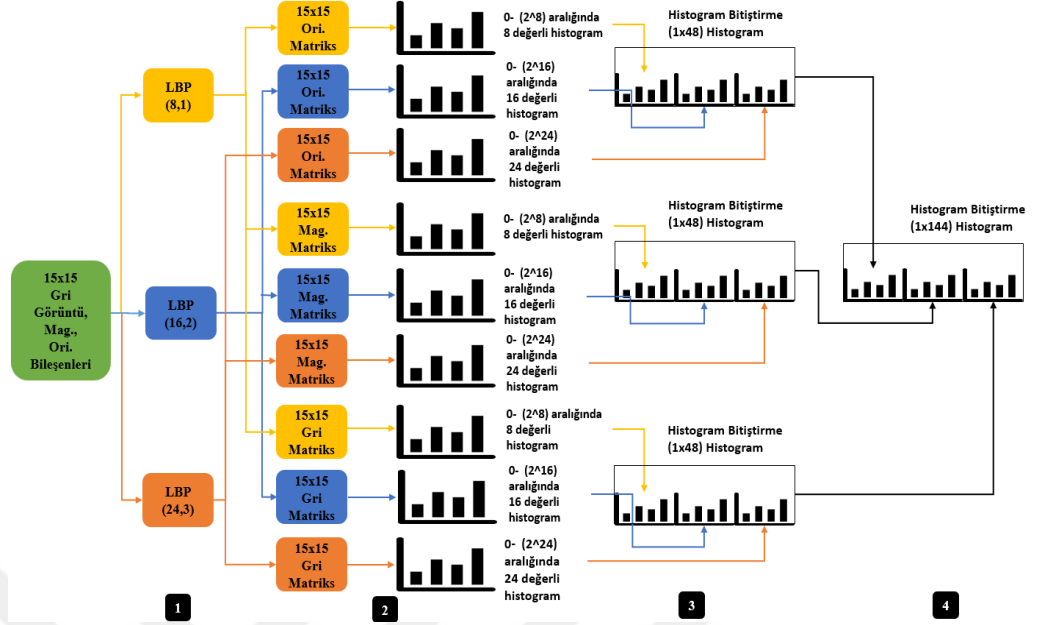
1. Köşe noktalarının tespiti,
2. Köşe noktalarına ait özniteliklerin oluşturulması,
3. Özniteliklerin eşlenmesi,
4. Aykırı noktaların tespiti ve elenmesi.



Şekil 4.2: Önerilen yöntem adımları

Şekil 4.2’de önerilen yöntem adımlarına ait yapı gösterilmektedir. Tez çalışması kapsamında yapılan kopyala taşı sahtecilik tespitinde ilk adım görüntünün gri seviye bileşenin çıkarılması ve köşe noktalarının tespitidir. Köşe noktalarının tespiti için birçok çalışmada kullanılan ve Lowe’nın 2003 yılında ortaya koyduğu çalışmasında ölçeksel değişimlere karşı dayanıklılığını ispat ettiği SIFT köşe nokta tespit algoritması kullanılmıştır. SIFT köşe tespit yöntemi ölçeksel değişimlere karşı dayanıklı ve diğer yöntemlere göre daha nitelikli noktalar tespit ettiğinden dolayı bu çalışmada tercih edilmiştir.

SIFT köşe noktaları tespit edildikten sonra görüntü Gauss filtresinden geçirilmektedir. Daha sonra SIFT ile tespit edilen her bir köşe noktasına ait öznitelikler oluşturulmaktadır. Öznitelik oluşturma işlemi için ilk olarak köşe noktaları etrafında belirli büyüklükte dörtgensel alanlar oluşturulmaktadır. Oluşturulan alanlar LBP algoritmasından geçirilmektedir. Bu işlem sonunda her bir köşe noktası için 144 birimlik öznitelik vektörü oluşturulmaktadır.



Şekil 4.3: Öznitelik çıkarma işlemleri

Şekil 4.3'te önerilen yöntem içerisinde kullanılan öznitelik çıkarma işlemine ait adımlar gösterilmektedir. Önerilen yöntemde köşe noktalarının tespitinden sonra ilk olarak köşe noktaları etrafında belirli büyüklükte dörtgensel alanlar oluşturulmaktadır. Daha sonra oluşturulan alanlar içerisindeki piksellere ait gri seviye, büyüklük ve oryantasyon değerleri ayrı ayrı hesaplanıp 3 farklı matris içine yerleştirilmektedir. Bir sonraki adımda oluşturulan 3 farklı matris (8,1), (16,2) ve (24,3) LBP operatörlerinden geçirilmektedir. Numaralandırılmış adımlara sırasıyla açıklanmaktadır.

1. Birinci adımda ilk olarak köşe noktasının öznitelik alanları ait farklı yerel ikili örüntü operatörlerinden geçirilip farklı bileşenleri çıkarılmaktadır. Önerilen yöntemde kullanılan yerel ikili örüntü operatörleri Ojala'nın çalışmasında önerdiği (8,1), (16,2) ve (24,3) operatörleri, bileşenler ise gri seviye dönüşümü, oryantasyon ve büyüklük olarak belirlenmiştir. İlk olarak köşe noktası etrafından çıkarılan 15x15'lik gri seviye matrise ait büyüklük ve oryantasyon matrisleri hesaplanmaktadır. Daha sonra gri seviye, büyüklük ve oryantasyon matrislerine ait yerel ikili örüntü değerleri hesaplanmaktadır. Bu işlem sonucunda toplamda 9 farklı matris hesaplanmaktadır.

2. İkinci adımda farklı bileşenler üzerinden hesaplanan yerel ikili örüntü matrisleri gruplandırılmaktadır. Yani oryantasyon için hesaplanan matrisler kendi aralarında, gri seviye matrisleri kendi aralarında ve büyüklük matrisleri kendi aralarında gruplandırılmaktadır. Bu şekilde 9 matris kendi arasında 3 farklı gruba bölünmektedir. Bir sonraki adımda her bir grup içerisinde bulunan matrislere ait histogramlar çıkarılmaktadır. Histogramlar çıkarılırken LBP operatörleri dikkate alınmaktadır. Açıklamak gerekirse (8,1) LBP matrisi 0-256 değer aralığına sahip olacaktır ve histogram hesaplaması yapılırken 0-256 değer aralığı 8 eş parçaya bölünerek yapılmıştır. Aynı şekilde (16,2) operatörüne ait histogram 16 eş parçaya bölünecek ve (24,3) histogramı ise 24 eş parçaya bölünecektir. Önerilen yöntemde histogram parça sayısının operatörün özelliğine göre değişim göstermektedir.
3. Üçüncü adımda ise farklı bileşenlere göre gruplaması yapılan histogramlar uç uca eklenmiştir. Bu şekilde her bir gruba ait 48 birim uzunluğunda histogram elde edilmiştir.
4. Dördüncü ve son adımda ise bir önceki adımda oluşturulan histogramlar tekrardan uç uca eklenmiş ve ilgi noktasına ait 144 birim uzunluğunda yeni bir histogram oluşturulmuştur. Bu adımda elde edilen histogram ileride VBSD eşleştirmesi için kullanılacaktır.

Özniteliklerin oluşturulmasından sonra öznitelikler eşlenmektedir. Önerilen yöntemde öznitelikler 2 tekrarlı VBSD algoritması ile eşlenmektedir. Kullanılan VBSD yönteminde histogramlar arası yakınlık ölçümü Manhattan uzaklığı ile hesaplanmıştır. Öznitelikler eşlenmesinden sonra son adımda yanlış eşleşmeler elenmektedir. Bu işlem için RANSAC algoritması kullanılmaktadır. RANSAC algoritması aykırı verileri elemek için kullanılan bir algoritmadır. RANSAC içindeki rastgele çalışma mekanizmasından dolayı her zaman aynı sonuçları vermemektedir (Dağdeviren, 2015)

Tez kapsamında hazırlanan algorithmada kullanılan parametreler deneysel çalışmalar sonucunda ortaya çıkan en iyi eşleme performansını sağlayan parametrelerdir. İlk olarak SIFT köşe nokta tespit yöntemi ile ölçeksel değişimlere dayanıklı köşe noktalarının tespit edilmesi sağlanmıştır. Görüntünün gri seviye bileşeninin çıkarılması ve Gauss filtresinden geçirilmesi sayesinde görüntü,

üzerinde gürültü ve ışık efektinden kaynaklı değişimlere karşı daha güçlü hale getirilmiştir. Öznitelik oluşumunda Ojala'nın 2002 yılında hazırlamış olduğu çalışmada belirttiği gibi farklı LBP operatörleri bir arada kullanılmış olup öznitelikler ölçeksel değişimlere karşı daha dayanıklı hale getirilmiştir. İki iterasyonlu VBSD eşleme yöntemi ile eşlenen öznitelikler döndürmeden ve ışık efektinden kaynaklı değişimlere karşı güçlü hale getirilmiştir. Önerilen yönteme ait kod parçası Algoritma 3.5'de gösterilmektedir.

*Başla*

***SIFT köşe noktalarını tespit et ve her bir köşe noktası için:***

*Adım 1: Köşe noktaları etrafında belirlenen büyüklükte dörtgensel alanı oluştur.*

*Adım 2: Dörtgensel alana ait gri seviye, oryantasyon ve büyüklük matrislerini oluştur.*

***Her bir matris için:***

*Adım 3: Matrisi belirlenen yerel ikili operatör algoritmasından geçir ve sonucu köşe noktasına ait yapıda sakla.*

*Adım 4: 3'te elde edilen 9 matrisi tiplerine göre gruplandır.*

*Adım 5: Aynı matris tiplerini yerel ikili operatör tiplerine göre sıralayarak uç uca ekle*

*Adım 6: Adım 5'te uç uca eklenen matrisleri tekrardan uç uca ekle ve 144 birimlik histogramı elde et.*

*Adım 7: 144 birimlik histogramları VBSD algoritması ile eşle.*

*Bitir*

Algoritma 3.5 Önerilen Yöntem sözde kodu

Şekil 4.4'te kopyala taşı sahtecilik tespiti sonuçları adım adım gösterilmektedir.



Şekil 4.4: COMOFOOD-26 Sonuçları a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Önerilen yöntem eşleme sonrası g) RANSAC sonrası

## 5.BULGULAR

Bu bölümde tez çalışması kapsamında hazırlanan algoritmaya ait deneysel sonuçlara ait detaylar belirtilmektedir. İlk olarak önerilen yöntem farklı parametreler ile COMOFOD veri setine bulunan 200 temel resim veri seti üzerinde test edilip en başarılı tespiti sunan parametreler elde edilmiştir. Daha sonra elde edilen bulgulardan yola çıkarak önerilen yöntem, COMOFOD ve CVIP Group Copy Move Forgery Dataset veri setleri üzerinde literatürdeki diğer algoritmalar ile kıyaslanmıştır.

COMOFOD veri setinde bulunan görüntüler farklı kopyala taşı sahtecilik türlerine sahiptirler. COMOFOD veri setinde bulunan kopyala taşı sahtecilik türlerine ait açıklama Tablo 5.1’de, tüm görüntülere ait detaylı açıklama Tablo 5.2’de belirtilmektedir.

Tablo 5.1: COMOFOD veri seti içerikleri

| Görüntü Aralıkları | İndeks | Tür       |
|--------------------|--------|-----------|
| [0,41]             |        | Öteleme   |
| [41,80]            |        | Döndürme  |
| [81,120]           |        | Ölçekleme |
| [121,159]          |        | Çarpıtma  |
| [160,200]          |        | Karma     |

Tablo 5.2: COMOFOD veri seti görüntü numaraları ve gerçekleştirilen bozulma türleri

| Ölçeklendirme Açıklamaları = Orijinal Boyut[İşlem Sonrası Boyut] Uygulanan Dönüşüm(R=Döndürme, S =Boyutlandırma, V=Dikey germe, H=Yatay germe, VS=Dikey eğme, HS=Yatay eğme) |                 |                  |                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|------------------|-----------------------|
| Görüntü Numarası                                                                                                                                                             | İşlem Açıklama  | Görüntü Numarası | İşlem Açıklama        |
| 1                                                                                                                                                                            | 2122 (Tek Alan) | 2                | 1470, 4158 (İki Alan) |
| 3                                                                                                                                                                            | 2256            | 4                | 998 (2) (İki Alan)    |
| 5                                                                                                                                                                            | 9852            | 6                | 19784                 |
| 7                                                                                                                                                                            | 1249, 1356      | 8                | 360                   |
| 9                                                                                                                                                                            | 3540            | 10               | 403                   |
| 11                                                                                                                                                                           | 1282            | 12               | 7380                  |
| 13                                                                                                                                                                           | 11873           | 14               | 2458                  |

|           |                                |           |                                        |
|-----------|--------------------------------|-----------|----------------------------------------|
| <b>15</b> | 14186                          | <b>16</b> | 2632                                   |
| <b>17</b> | 3927                           | <b>18</b> | 1122                                   |
| <b>19</b> | 1122, 701, 411                 | <b>20</b> | 701 (3)                                |
| <b>21</b> | 2536                           | <b>22</b> | 841 (2)                                |
| <b>23</b> | 2624, 415, 183                 | <b>24</b> | 28405                                  |
| <b>25</b> | 1589 (2)                       | <b>26</b> | 3369                                   |
| <b>27</b> | 4340 (2)                       | <b>28</b> | 15478                                  |
| <b>29</b> | 10376                          | <b>30</b> | 4928 (2)                               |
| <b>31</b> | 1651                           | <b>32</b> | 2410                                   |
| <b>33</b> | 3564                           | <b>34</b> | 13683                                  |
| <b>35</b> | 37208                          | <b>36</b> | 1316                                   |
| <b>37</b> | 726                            | <b>38</b> | 2921 (2)                               |
| <b>39</b> | 2122, 1037                     | <b>40</b> | 22865                                  |
| <b>41</b> | 998[998] R180                  | <b>42</b> | 998[998] R180, 998[998] R180           |
| <b>43</b> | 3540[3647] R10                 | <b>44</b> | 3540[3654] R2                          |
| <b>45</b> | 403[419] R2                    | <b>46</b> | 1282[1282] R180                        |
| <b>47</b> | 1282[1282] R90                 | <b>48</b> | 11873[12196] R2                        |
| <b>49</b> | 2458[2576] R2                  | <b>50</b> | 14186[14490] R2                        |
| <b>51</b> | 14186[14690] R4                | <b>52</b> | 14186[14492] R-4                       |
| <b>53</b> | 3927[4061] R5                  | <b>54</b> | 3927[4055] R-7, 3927[4061] R5          |
| <b>55</b> | 701[800] R3                    | <b>56</b> | 701[798] R5, 701[800] R-3, 701[800] R3 |
| <b>57</b> | 1122[1236] R-80                | <b>58</b> | 2536[2666] R2                          |
| <b>59</b> | 3369[3401] R7                  | <b>60</b> | 2107[2277] R10                         |
| <b>61</b> | 15478[15838] R40               | <b>62</b> | 10376[10571] R50                       |
| <b>63</b> | 2632[2632] R180                | <b>64</b> | 37542[37601] R2                        |
| <b>65</b> | 5264[5287] R5                  | <b>66</b> | 1241[1317] R10, 1241[1324] R30         |
| <b>67</b> | 1079[1068] R2, 1110[1110] R180 | <b>68</b> | 7611[7611] R180                        |
| <b>69</b> | 3616[3602] R3                  | <b>70</b> | 1037[1024] R-3                         |
| <b>71</b> | 1651[1707] R3                  | <b>72</b> | 2449[2410] R2                          |
| <b>73</b> | 37208[37413] R-1               | <b>74</b> | 18731[18796] R4                        |
| <b>75</b> | 22865[23220] R-3               | <b>76</b> | 726[726] R-1                           |
| <b>77</b> | 1316[1409] R2                  | <b>78</b> | 3369[3407] R2, 3369[3403] R3           |
| <b>79</b> | 2921[3034] R-1                 | <b>80</b> | 1589[1597] R2                          |

|            |                                    |            |                                       |
|------------|------------------------------------|------------|---------------------------------------|
| <b>81</b>  | 1282[1141] S90                     | <b>82</b>  | 3375[1768] S70                        |
| <b>83</b>  | 3375[1768] S70,<br>1282[1141] S90  | <b>84</b>  | 1282[1629] S110                       |
| <b>85</b>  | 9852[10318] S101                   | <b>86</b>  | 3540[6006] S130                       |
| <b>87</b>  | 3540[2256] S80                     | <b>88</b>  | 3540[2256]S80,<br>3540[6006]S130      |
| <b>89</b>  | 403[382] S95                       | <b>90</b>  | 11873[13471] S105                     |
| <b>91</b>  | 2458[2089] S90                     | <b>92</b>  | 2458[2689] S102                       |
| <b>93</b>  | 3927[3060] S87                     | <b>94</b>  | 2536[3504] S115                       |
| <b>95</b>  | 701[1082] S117                     | <b>96</b>  | 701[320] S60 701[1333] S130           |
| <b>97</b>  | 998[1142] S105                     | <b>98</b>  | 14186[7180] S70                       |
| <b>99</b>  | 5316[3844] S85 *                   | <b>100</b> | 1589[2600] S130                       |
| <b>101</b> | 3369[4501] S115                    | <b>102</b> | 4340[1649] S60                        |
| <b>103</b> | 15478[2620] S40                    | <b>104</b> | 10376[20583] S150                     |
| <b>105</b> | 3510[891] S50                      | <b>106</b> | 37542[15878] S65                      |
| <b>107</b> | 5264[3998] S87,<br>5264[5858] S105 | <b>108</b> | 1241[1896] S120, 4124[2145]<br>S70    |
| <b>109</b> | 8640[3290] S60                     | <b>110</b> | 2122[2937] S115                       |
| <b>111</b> | 1651[1864] S105                    | <b>112</b> | 2410[2231] S95                        |
| <b>113</b> | 3564[4061] S107                    | <b>114</b> | 13683[13229] S98                      |
| <b>115</b> | 37208[38878] S102                  | <b>116</b> | 1316[1288] S98                        |
| <b>117</b> | 22865[16717] s85                   | <b>118</b> | 2921[2967] S102 (2)                   |
| <b>119</b> | 2122[1996] S95,<br>1037[944] S97   | <b>120</b> | 18731[17279] S96                      |
| <b>121</b> | 2122[2348] V110                    | <b>122</b> | 3375[3288] V95                        |
| <b>123</b> | 2256[2736] V120                    | <b>124</b> | 3540[2832] V80                        |
| <b>125</b> | 2458[2289] V90,<br>2458[2658] V105 | <b>126</b> | 14186[16517] V115                     |
| <b>127</b> | 3927[3451] V85                     | <b>128</b> | 3369[4219] V125                       |
| <b>129</b> | 37542[28137] V75                   | <b>130</b> | 10376[10973] V105                     |
| <b>131</b> | 2122[2542] H115                    | <b>132</b> | 3375[2780] H80                        |
| <b>133</b> | 2256[2068] H90                     | <b>134</b> | 3540[3957] H110                       |
| <b>135</b> | 2458[2634] H105                    | <b>136</b> | 14186[13661] H95                      |
| <b>137</b> | 3927[4761] H120                    | <b>138</b> | 3369[3549] H105                       |
| <b>139</b> | 37542[35737] H95                   | <b>140</b> | 10376[11467] H110,<br>10376[8877] H85 |
| <b>141</b> | 1037[1034] VS5                     | <b>142</b> | 3375[3462] VS8                        |
| <b>143</b> | 2256[2304] VS2                     | <b>144</b> | 3540[3597] VS3                        |

|            |                                          |            |                                               |
|------------|------------------------------------------|------------|-----------------------------------------------|
| <b>145</b> | 2458[2533] VS4                           | <b>146</b> | 14186[14362] VS1                              |
| <b>147</b> | 3927[4002] VS6                           | <b>148</b> | 3369[3378] VS3                                |
| <b>149</b> | 37542[37545] VS4                         | <b>150</b> | 10376[10464] VS3                              |
| <b>151</b> | 1037[1028] HS3                           | <b>152</b> | 3375[3450] HS7                                |
| <b>153</b> | 2256[2300] HS5                           | <b>154</b> | 3540[3597] HS2                                |
| <b>155</b> | 2458[2513] HS6                           | <b>156</b> | 14186[14344] HS4                              |
| <b>157</b> | 3927[3983] HS1                           | <b>158</b> | 3369[3395] HS2                                |
| <b>159</b> | 37542[37570] HS5                         | <b>160</b> | 10376[10433] HS8,<br>10376[10432] HS3         |
| <b>161</b> | 1282[2307] S150 R90                      | <b>162</b> | 14186[7104] S70 R3                            |
| <b>163</b> | 3927[3293] S90 R-7                       | <b>164</b> | 2536[3603] S115 R2                            |
| <b>165</b> | 403[407] S95 R-5                         | <b>166</b> | 2458[2134] S90 R1                             |
| <b>167</b> | 11873[13733] S105 R-2                    | <b>168</b> | 3540[2382] S80 R45                            |
| <b>169</b> | 1122[1939] S125 R-80                     | <b>170</b> | 998[1142] S105 R90                            |
| <b>171</b> | 21282[10940] S70 R-1                     | <b>172</b> | 3616[1774] S70 R5                             |
| <b>173</b> | 1241[1745] S115 R5                       | <b>174</b> | 15478[11538] S85 R45                          |
| <b>175</b> | 3369[3741] S105 R3                       | <b>176</b> | 2107[3288] S120 R3                            |
| <b>177</b> | 998[1205] S105 R45                       | <b>178</b> | 1350[1430] S102 R1                            |
| <b>179</b> | 1079[979] S95 R2,<br>1110[1336] S110 R90 | <b>180</b> | 37542[27162] S85 R2                           |
| <b>181</b> | 14186[17100] V118 R3                     | <b>182</b> | 2122[2390] H109 HS2                           |
| <b>183</b> | 3369[3539] V105 VS3                      | <b>184</b> | 27255[28665] V103 H101                        |
| <b>185</b> | 10376[10480] VS5 HS3                     | <b>186</b> | 3505[3307] H95 VS4                            |
| <b>187</b> | 8640[2186] H60 V40 HS6                   | <b>188</b> | 1589[1650] V105 VS1                           |
| <b>189</b> | 2624[2778] V103 VS1                      | <b>190</b> | 2122[2537] V110 H103 VS2,<br>1037[978] H95 R3 |
| <b>191</b> | 1651[1812] R2 S103                       | <b>192</b> | 2410[2324] R2 S97                             |
| <b>193</b> | 37208[38911] R-1 S102                    | <b>194</b> | 18731[15226] S90 R3                           |
| <b>195</b> | 22865[16778] S85 R3                      | <b>196</b> | 14186[15810] H109 R3                          |
| <b>197</b> | 1316[1465] S102 R2                       | <b>198</b> | 1651[1746] V105 VS2                           |
| <b>199</b> | 726[778] S103 R2,<br>338[338] S95 R2     | <b>200</b> | 3375[3623] H103 VS2,<br>3375[3355] V95 HS3    |

Tablo 5.1’de belirtilen türler temel alındığında önerilen yöntemin başarısı öteleme, döndürme, ölçek değişimi, çarpıtma ve tüm atakların birleşimi olmak üzere beş farklı alanda değerlendirilmektedir.

Tez çalışması kapsamında algoritmaların başarı ölçütleri karşılaştırılmasında COMOFOD veri setinin yanı sıra CVIP veri seti kullanılmıştır. Veri set içerisinde farklı türlerde iki alt veri seti bulunmaktadır. İlk alt veri setinde 50, ikinci alt veri setinden 550 adet görüntü algoritmaların performans karşılaştırması için tercih edilmiştir. Tablo 5.3'te CVIP veri setinde kullanılan görüntülere ait sayısal bilgilere yer verilmektedir.

Tablo 5.3: CVIP veri seti içerikleri

| CVIP Alt Veri Seti | Adet | Tür       | Değer |
|--------------------|------|-----------|-------|
| Veri seti 1        | 50   | Öteleme   |       |
| Veri seti 2        | 20   | Döndürme  | -15   |
| Veri seti 2        | 20   | Döndürme  | -10   |
| Veri seti 2        | 20   | Döndürme  | -5    |
| Veri seti 2        | 20   | Döndürme  | -4    |
| Veri seti 2        | 20   | Döndürme  | -3    |
| Veri seti 2        | 20   | Döndürme  | -2    |
| Veri seti 2        | 20   | Döndürme  | -1    |
| Veri seti 2        | 20   | Döndürme  | 0     |
| Veri seti 2        | 20   | Döndürme  | 1     |
| Veri seti 2        | 20   | Döndürme  | 2     |
| Veri seti 2        | 20   | Döndürme  | 3     |
| Veri seti 2        | 20   | Döndürme  | 4     |
| Veri seti 2        | 20   | Döndürme  | 5     |
| Veri seti 2        | 20   | Döndürme  | 10    |
| Veri seti 2        | 20   | Döndürme  | 15    |
| Veri seti 2        | 20   | Ölçekleme | 0.75  |
| Veri seti 2        | 20   | Ölçekleme | 0.80  |
| Veri seti 2        | 20   | Ölçekleme | 0.85  |
| Veri seti 2        | 20   | Ölçekleme | 0.90  |
| Veri seti 2        | 20   | Ölçekleme | 0.95  |
| Veri seti 2        | 20   | Ölçekleme | 1.05  |
| Veri seti 2        | 20   | Ölçekleme | 1.1   |
| Veri seti 2        | 20   | Ölçekleme | 1.15  |
| Veri seti 2        | 20   | Ölçekleme | 1.2   |
| Veri seti 2        | 20   | Ölçekleme | 1.25  |

Tez çalışması kapsamında performans değerlendirmeleri için doğruluk (*ing.* accuracy), kesinlik (*ing.* precision) ve yanlış tahmin oranı (FPR) metrikleri kullanılmıştır. Denklem 5.1,5.2, 5.3'te metriklere ait hesaplama belirtilmektedir.

Metrik hesaplamasında kullanılan TP (Doğru Pozitif) değeri doğru tespit edilen kopyalanmış nokta sayısını, FP (Yanlış Pozitif) kopyalanmış olarak işaretlenen normal nokta sayısını, TN (Doğru Negatif) normal olarak işaretlenen normal nokta sayısını, FN (Yanlış Negatif) normal olarak işaretlenen kopyalanmış nokta sayısını belirtmektedir. Yapılan hesaplamalar görüntüler üzerindeki algoritma çıktısında elde edilen pikseller ve COMOFOOD veri seti sonuçları üzerindeki piksellerin karşılaştırmaları kullanılarak elde edilmiştir. Denklem 5.1-5.3'te karşılaştırmada kullanılan değerlere ait hesaplamalar gösterilmektedir.

$$Doğruluk = \frac{T_p + T_N}{T_p + F_p + T_N + F_N} \quad (5.1)$$

$$Kesinlik = \frac{T_p}{T_p + F_p} \quad (5.2)$$

$$FPR = \frac{F_p}{F_p + T_N} \quad (5.3)$$

Tez çalışması kapsamında ilk olarak önerilen yönteme ait en etkili eşleme performansı veren parametreler tespit edilmiştir. Tablo 5.4 – 5.7'te detaylı olarak farklı parametreler ile elde edilen eşleme performanslarına ait değerler gösterilmektedir.

Tablo 5.4: Önerilen yöntem (Ö.Y) 11x11 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD),  
Manhattan Uzaklığı (MD) test sonuçları

| Atak      | E. Yöntem                                                                              | VBSD İ.S. | TP    | FP    | FN    | TN     | KTS N.S. | Doğru luk    | Kesinlik     | FPR          |
|-----------|----------------------------------------------------------------------------------------|-----------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | <b>11x11 öznitelik alan büyüklüğü, 232314 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 13170 | 3396  | 4017  | 211731 | 16569    | 0.968        | 0.795        | 0.015        |
|           | JD                                                                                     | 2         | 13926 | 16611 | 3261  | 198516 | 30540    | 0.914        | 0.456        | 0.077        |
|           | MD                                                                                     | 2         | 13455 | 2937  | 3732  | 212190 | 16392    | <b>0.971</b> | <b>0.820</b> | <b>0.013</b> |
|           | Öklit                                                                                  | 3         | 13131 | 3585  | 4056  | 211542 | 16716    | 0.967        | 0.785        | 0.016        |
|           | JD                                                                                     | 3         | 13881 | 17670 | 3306  | 197457 | 31560    | 0.909        | 0.439        | 0.082        |
|           | MD                                                                                     | 3         | 13374 | 3306  | 3813  | 211821 | 16680    | 0.969        | 0.801        | 0.015        |
| Döndürme  | <b>11x11 öznitelik alan büyüklüğü, 277560 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 6141  | 2247  | 11952 | 257223 | 8388     | 0.948        | 0.732        | 0.008        |
|           | JD                                                                                     | 2         | 8535  | 13920 | 9561  | 245550 | 22455    | 0.915        | 0.380        | 0.053        |
|           | MD                                                                                     | 2         | 6003  | 1731  | 12090 | 257739 | 7734     | <b>0.950</b> | <b>0.776</b> | <b>0.006</b> |
|           | Öklit                                                                                  | 3         | 5967  | 2298  | 12126 | 257172 | 8265     | 0.948        | 0.721        | 0.008        |
|           | JD                                                                                     | 3         | 8319  | 15297 | 9777  | 244173 | 23619    | 0.909        | 0.352        | 0.058        |
|           | MD                                                                                     | 3         | 5943  | 1998  | 12150 | 257472 | 7941     | 0.949        | 0.748        | 0.007        |
| Ölçekleme | <b>11x11 öznitelik alan büyüklüğü, 268104 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4296  | 2511  | 14775 | 246516 | 6807     | 0.935        | 0.631        | 0.010        |
|           | JD                                                                                     | 2         | 5907  | 15363 | 13164 | 233664 | 21270    | 0.893        | 0.277        | 0.061        |
|           | MD                                                                                     | 2         | 4176  | 2085  | 14895 | 246942 | 6261     | <b>0.936</b> | <b>0.666</b> | <b>0.008</b> |
|           | Öklit                                                                                  | 3         | 4167  | 2733  | 14904 | 246294 | 6900     | 0.934        | 0.603        | 0.010        |
|           | JD                                                                                     | 3         | 5862  | 16494 | 13212 | 232533 | 22362    | 0.889        | 0.262        | 0.066        |
|           | MD                                                                                     | 3         | 4128  | 2436  | 14943 | 246591 | 6564     | 0.935        | 0.628        | 0.009        |
| Çarpıtma  | <b>11x11 öznitelik alan büyüklüğü, 278925 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12723 | 2091  | 13797 | 250314 | 14814    | 0.943        | 0.858        | 0.008        |
|           | JD                                                                                     | 2         | 15825 | 14136 | 10698 | 238269 | 29961    | 0.910        | 0.528        | 0.056        |
|           | MD                                                                                     | 2         | 12624 | 1908  | 13896 | 250497 | 14532    | <b>0.943</b> | <b>0.868</b> | <b>0.007</b> |
|           | Öklit                                                                                  | 3         | 12285 | 2268  | 14235 | 250137 | 14553    | 0.940        | 0.844        | 0.008        |
|           | JD                                                                                     | 3         | 15381 | 15687 | 11142 | 236718 | 31068    | 0.903        | 0.495        | 0.062        |
|           | MD                                                                                     | 3         | 12450 | 2106  | 14070 | 250299 | 14556    | 0.942        | 0.855        | 0.008        |
| Karma     | <b>11x11 öznitelik alan büyüklüğü, 279210 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 5037  | 3117  | 16776 | 254280 | 8154     | 0.928        | 0.617        | 0.012        |
|           | JD                                                                                     | 2         | 8049  | 15498 | 13767 | 241899 | 23547    | 0.895        | 0.341        | 0.060        |
|           | MD                                                                                     | 2         | 4959  | 2493  | 16854 | 254904 | 7452     | <b>0.930</b> | <b>0.665</b> | <b>0.009</b> |
|           | Öklit                                                                                  | 3         | 4899  | 3051  | 16914 | 254346 | 7950     | 0.928        | 0.616        | 0.011        |
|           | JD                                                                                     | 3         | 7677  | 16773 | 14139 | 240624 | 24450    | 0.889        | 0.313        | 0.065        |
|           | MD                                                                                     | 3         | 4830  | 2646  | 16983 | 254751 | 7476     | 0.929        | 0.646        | 0.010        |

Tablo 5.5: Önerilen yöntem (Ö.Y) 13x13 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD),  
Manhattan Uzaklığı (MD) test sonuçları

| Atak      | E. Yöntem                                                                              | VBSD İ.S. | TP    | FP    | FN    | TN     | KTS N.S. | Doğruluk     | Kesinlik     | FPR          |
|-----------|----------------------------------------------------------------------------------------|-----------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | <b>13x13 öznitelik alan büyüklüğü, 232314 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12918 | 3387  | 4269  | 211740 | 16305    | 0.967        | 0.792        | 0.015        |
|           | JD                                                                                     | 2         | 13581 | 14190 | 3606  | 200937 | 27771    | 0.923        | 0.489        | 0.065        |
|           | MD                                                                                     | 2         | 13092 | 2712  | 4095  | 212415 | 15804    | <b>0.970</b> | <b>0.828</b> | <b>0.012</b> |
|           | Öklit                                                                                  | 3         | 12831 | 3618  | 4356  | 211509 | 16449    | 0.965        | 0.780        | 0.016        |
|           | JD                                                                                     | 3         | 13515 | 15015 | 3672  | 200112 | 28530    | 0.919        | 0.473        | 0.069        |
|           | MD                                                                                     | 3         | 13005 | 2922  | 4182  | 212205 | 15927    | 0.969        | 0.816        | 0.013        |
| Döndürme  | <b>13x13 öznitelik alan büyüklüğü, 277560 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 6264  | 2151  | 11832 | 257319 | 8415     | 0.949        | 0.744        | 0.008        |
|           | JD                                                                                     | 2         | 8313  | 12357 | 9783  | 247113 | 20670    | 0.920        | 0.402        | 0.047        |
|           | MD                                                                                     | 2         | 6042  | 1602  | 12054 | 257868 | 7644     | <b>0.950</b> | <b>0.790</b> | <b>0.006</b> |
|           | Öklit                                                                                  | 3         | 6102  | 2439  | 11991 | 257031 | 8541     | 0.948        | 0.714        | 0.009        |
|           | JD                                                                                     | 3         | 8001  | 13671 | 10095 | 245799 | 21672    | 0.914        | 0.369        | 0.052        |
|           | MD                                                                                     | 3         | 5946  | 1707  | 12147 | 257763 | 7653     | 0.950        | 0.776        | 0.006        |
| Ölçekleme | <b>13x13 öznitelik alan büyüklüğü, 268104 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4248  | 2604  | 14823 | 246423 | 6852     | 0.934        | 0.619        | 0.010        |
|           | JD                                                                                     | 2         | 5577  | 13275 | 13494 | 235752 | 18852    | 0.900        | 0.295        | 0.053        |
|           | MD                                                                                     | 2         | 4131  | 2001  | 14940 | 247026 | 6132     | <b>0.936</b> | <b>0.673</b> | <b>0.008</b> |
|           | Öklit                                                                                  | 3         | 4212  | 2886  | 14859 | 246141 | 7098     | 0.933        | 0.593        | 0.011        |
|           | JD                                                                                     | 3         | 5490  | 14682 | 13581 | 234345 | 20172    | 0.894        | 0.272        | 0.058        |
|           | MD                                                                                     | 3         | 4083  | 2103  | 14988 | 246924 | 6186     | 0.936        | 0.660        | 0.008        |
| Çarpıtma  | <b>13x13 öznitelik alan büyüklüğü, 278925 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12753 | 2226  | 13767 | 250179 | 14979    | 0.942        | 0.851        | 0.008        |
|           | JD                                                                                     | 2         | 15408 | 11712 | 11112 | 240693 | 27120    | 0.918        | 0.568        | 0.046        |
|           | MD                                                                                     | 2         | 12474 | 1623  | 14046 | 250782 | 14097    | <b>0.943</b> | <b>0.884</b> | <b>0.006</b> |
|           | Öklit                                                                                  | 3         | 12420 | 2523  | 14100 | 249882 | 14943    | 0.940        | 0.831        | 0.009        |
|           | JD                                                                                     | 3         | 14826 | 13527 | 11697 | 238878 | 28353    | 0.909        | 0.522        | 0.053        |
|           | MD                                                                                     | 3         | 12327 | 1596  | 14193 | 250809 | 13923    | 0.943        | 0.885        | 0.006        |
| Karma     | <b>13x13 öznitelik alan büyüklüğü, 279210 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4968  | 2907  | 16845 | 254490 | 7875     | 0.929        | 0.630        | 0.011        |
|           | JD                                                                                     | 2         | 7491  | 13701 | 14322 | 243696 | 21195    | 0.899        | 0.353        | 0.053        |
|           | MD                                                                                     | 2         | 4689  | 2274  | 17124 | 255123 | 6963     | <b>0.930</b> | <b>0.673</b> | <b>0.008</b> |
|           | Öklit                                                                                  | 3         | 4914  | 3234  | 16899 | 254163 | 8148     | 0.927        | 0.603        | 0.012        |
|           | JD                                                                                     | 3         | 7086  | 14799 | 14727 | 242598 | 21888    | 0.894        | 0.323        | 0.057        |
|           | MD                                                                                     | 3         | 4743  | 2364  | 17070 | 255033 | 7107     | 0.930        | 0.667        | 0.009        |

Tablo 5.6: Önerilen yöntem (Ö.Y) 15x15 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD),  
Manhattan Uzaklığı (MD) test sonuçları

| Atak      | E. Yöntem                                                                              | VBSD İ.S. | TP    | FP    | FN    | TN     | KTS N.S. | Doğru luk    | Kesinlik     | FPR          |
|-----------|----------------------------------------------------------------------------------------|-----------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | <b>15x15 öznitelik alan büyüklüğü, 232314 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12672 | 3441  | 4515  | 211686 | 16113    | 0.965        | 0.786        | 0.015        |
|           | JD                                                                                     | 2         | 13335 | 12846 | 3852  | 202281 | 26184    | 0.928        | 0.509        | 0.059        |
|           | MD                                                                                     | 2         | 12885 | 2484  | 4302  | 212643 | 15369    | <b>0.970</b> | <b>0.838</b> | <b>0.011</b> |
|           | Öklit                                                                                  | 3         | 12525 | 3597  | 4662  | 211530 | 16122    | 0.964        | 0.776        | 0.016        |
|           | JD                                                                                     | 3         | 13173 | 13644 | 4014  | 201483 | 26817    | 0.923        | 0.491        | 0.063        |
|           | MD                                                                                     | 3         | 12741 | 2790  | 4446  | 212337 | 15531    | 0.968        | 0.820        | 0.012        |
| Döndürme  | <b>15x15 öznitelik alan büyüklüğü, 277560 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 6360  | 2433  | 11730 | 257037 | 8793     | 0.948        | 0.723        | 0.009        |
|           | JD                                                                                     | 2         | 7932  | 11220 | 10161 | 248250 | 19152    | 0.922        | 0.414        | 0.043        |
|           | MD                                                                                     | 2         | 5985  | 1659  | 12105 | 257811 | 7644     | <b>0.950</b> | <b>0.782</b> | <b>0.006</b> |
|           | Öklit                                                                                  | 3         | 6213  | 2508  | 11877 | 256962 | 8721     | 0.948        | 0.712        | 0.009        |
|           | JD                                                                                     | 3         | 7815  | 12198 | 10275 | 247272 | 20013    | 0.919        | 0.390        | 0.047        |
|           | MD                                                                                     | 3         | 5991  | 1866  | 12099 | 257604 | 7857     | 0.949        | 0.762        | 0.007        |
| Ölçekleme | <b>15x15 öznitelik alan büyüklüğü, 268104 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4311  | 2616  | 14760 | 246411 | 6927     | 0.935        | 0.622        | 0.010        |
|           | JD                                                                                     | 2         | 5277  | 12033 | 13794 | 236994 | 17310    | 0.903        | 0.304        | 0.048        |
|           | MD                                                                                     | 2         | 4044  | 1920  | 15027 | 247107 | 5964     | <b>0.936</b> | <b>0.678</b> | <b>0.007</b> |
|           | Öklit                                                                                  | 3         | 4209  | 2898  | 14862 | 246129 | 7107     | 0.933        | 0.592        | 0.011        |
|           | JD                                                                                     | 3         | 5232  | 13275 | 13839 | 235752 | 18507    | 0.898        | 0.282        | 0.053        |
|           | MD                                                                                     | 3         | 3978  | 2271  | 15093 | 246756 | 6249     | 0.935        | 0.636        | 0.009        |
| Çarpıtma  | <b>15x15 öznitelik alan büyüklüğü, 278925 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12438 | 1953  | 14082 | 250452 | 14391    | 0.942        | 0.864        | 0.007        |
|           | JD                                                                                     | 2         | 14868 | 10917 | 11655 | 241488 | 25785    | 0.919        | 0.576        | 0.043        |
|           | MD                                                                                     | 2         | 12021 | 1203  | 14499 | 251202 | 13224    | <b>0.943</b> | <b>0.909</b> | <b>0.004</b> |
|           | Öklit                                                                                  | 3         | 11973 | 2007  | 14550 | 250398 | 13980    | 0.940        | 0.856        | 0.007        |
|           | JD                                                                                     | 3         | 14418 | 12123 | 12105 | 240282 | 26541    | 0.913        | 0.543        | 0.048        |
|           | MD                                                                                     | 3         | 11886 | 1440  | 14637 | 250965 | 13326    | 0.942        | 0.891        | 0.005        |
| Karma     | <b>15x15 öznitelik alan büyüklüğü, 279210 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4935  | 2958  | 16878 | 254439 | 7893     | 0.928        | 0.625        | 0.011        |
|           | JD                                                                                     | 2         | 6876  | 12381 | 14937 | 245016 | 19260    | 0.902        | 0.357        | 0.048        |
|           | MD                                                                                     | 2         | 4455  | 2151  | 17358 | 255246 | 6606     | <b>0.930</b> | <b>0.674</b> | <b>0.008</b> |
|           | Öklit                                                                                  | 3         | 4830  | 3129  | 16983 | 254268 | 7962     | 0.927        | 0.606        | 0.012        |
|           | JD                                                                                     | 3         | 6771  | 13491 | 15042 | 243906 | 20265    | 0.897        | 0.334        | 0.052        |
|           | MD                                                                                     | 3         | 4497  | 2361  | 17316 | 255036 | 6858     | 0.929        | 0.655        | 0.009        |

Tablo 5.7: Önerilen yöntem (Ö.Y) 17x17 alan büyüklüğü Öklit, Jeffrey Diverjansı (JD),  
Manhattan Uzaklığı (MD) test sonuçları

| Atak      | E. Yöntem                                                                              | VBSD İ.S. | TP    | FP    | FN    | TN     | KTS N.S. | Doğru luk    | Kesinlik     | FPR          |
|-----------|----------------------------------------------------------------------------------------|-----------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | <b>17x17 öznitelik alan büyüklüğü, 232314 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 12408 | 3363  | 4779  | 211764 | 15774    | 0.964        | 0.786        | 0.015        |
|           | JD                                                                                     | 2         | 13131 | 11469 | 4056  | 203658 | 24603    | 0.933        | 0.533        | 0.053        |
|           | MD                                                                                     | 2         | 12702 | 2343  | 4485  | 212784 | 15045    | <b>0.970</b> | <b>0.844</b> | <b>0.010</b> |
|           | Öklit                                                                                  | 3         | 12321 | 3519  | 4866  | 211608 | 15843    | 0.963        | 0.777        | 0.016        |
| Döndürme  | <b>17x17 öznitelik alan büyüklüğü, 277560 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 6306  | 2418  | 11790 | 257052 | 8727     | 0.948        | 0.722        | 0.009        |
|           | JD                                                                                     | 2         | 7767  | 10116 | 10329 | 249354 | 17886    | 0.926        | 0.434        | 0.038        |
|           | MD                                                                                     | 2         | 5922  | 1545  | 12174 | 257925 | 7467     | <b>0.950</b> | <b>0.793</b> | <b>0.005</b> |
|           | Öklit                                                                                  | 3         | 6162  | 2556  | 11934 | 256914 | 8721     | 0.947        | 0.706        | 0.009        |
| Ölçekleme | <b>17x17 öznitelik alan büyüklüğü, 268104 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4035  | 2856  | 15036 | 246171 | 6894     | 0.933        | 0.585        | 0.011        |
|           | JD                                                                                     | 2         | 4995  | 10953 | 14076 | 238074 | 15951    | 0.906        | 0.313        | 0.043        |
|           | MD                                                                                     | 2         | 3810  | 1872  | 15261 | 247155 | 5682     | <b>0.936</b> | <b>0.670</b> | <b>0.007</b> |
|           | Öklit                                                                                  | 3         | 3954  | 2880  | 15117 | 246147 | 6837     | 0.932        | 0.578        | 0.011        |
| Çarpıtma  | <b>17x17 öznitelik alan büyüklüğü, 278925 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 11949 | 1782  | 14571 | 250623 | 13731    | 0.941        | 0.870        | 0.007        |
|           | JD                                                                                     | 2         | 14298 | 9189  | 12222 | 243216 | 23487    | 0.923        | 0.608        | 0.036        |
|           | MD                                                                                     | 2         | 11703 | 1017  | 14817 | 251388 | 12720    | <b>0.943</b> | <b>0.920</b> | <b>0.004</b> |
|           | Öklit                                                                                  | 3         | 11838 | 2007  | 14682 | 250398 | 13845    | 0.940        | 0.855        | 0.007        |
| Karma     | <b>17x17 öznitelik alan büyüklüğü, 279210 adet köşe nokta sayısı deneysel sonuçlar</b> |           |       |       |       |        |          |              |              |              |
|           | Öklit                                                                                  | 2         | 4722  | 3153  | 17091 | 254244 | 7881     | 0.927        | 0.599        | 0.012        |
|           | JD                                                                                     | 2         | 6327  | 10956 | 15486 | 246441 | 17289    | 0.905        | 0.366        | 0.042        |
|           | MD                                                                                     | 2         | 4293  | 2085  | 17520 | 255312 | 6381     | <b>0.929</b> | <b>0.673</b> | <b>0.008</b> |
|           | Öklit                                                                                  | 3         | 4557  | 3186  | 17256 | 254211 | 7749     | 0.926        | 0.588        | 0.012        |

Tablo 5.4-5.7’te önerilen yöntemin COMOFOD veri seti üzerinde farklı parametreler ile test edilmesine ait sonuçlar gösterilmektedir. Tablolar dörtgensel alan büyüklüğü, ataklar, eşleşme yöntemi ve VBSD iterasyon sayısına göre farklılık göstermektedir. Tablolar içerisinde her atak öncesi başlık olarak alan büyüklükleri ve veri seti üzerinde bulunan resimler üzerinde kaçar adet SIFT köşe noktası olduğu başlık olarak belirtilmektedir. Önerilen yöntemle ait deneysel sonuçlar elde edilirken farklı alan büyüklükleri ve VBSD iterasyon sayısı değerleri alınmış olup üçüncü bölümde bahsedilen her bir benzerlik formülü ile birlikte kullanılmıştır.

Tez çalışmasında seçilen pencere büyüklük değerlerinin literatürdeki diğer algoritmalarından daha büyük olmaması gerekçesiyle maksimum 15x15 büyüklük alanı belirlenmiştir. Bu bağlamda yapılan deneyler sonucunda 15x15 öznitelik

büyüklik alanı, VBSD iterasyon sayısının 2 olarak belirlenmesi ve histogram benzerlik hesabında Manhattan uzaklığının kullanılması durumunda algoritmanın en etkili eşleme performansını gösterdiği gözlemlenmiştir. Tez kapsamında yapılan deneylerde bir sonraki adımda, ilk adımda belirlenen parametreler orjinal LBP algoritmaları ile karşılaştırılmıştır. Bu sayede algoritmanın orjinal LBP algoritmalarına göre üstün veya zayıf olduğu noktalar tespit edilmeye çalışılmıştır. Tablo 5.8’de hazırlanan algoritmanın LBP algoritmaları ile karşılaştırılmasına ait sonuçlar gösterilmektedir.

Tablo 5.8: Önerilen yöntem (Ö.Y.) ve orjinal LBP algoritmaları karşılaştırılması

| Atak      | Yöntem     | TP    | FP    | FN    | TN     | KTS N.S. | Doğruluk     | Kesinlik     | FPR          |
|-----------|------------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | Ö.Y.       | 12885 | 2484  | 4302  | 212643 | 15369    | <b>0.970</b> | <b>0.838</b> | <b>0.011</b> |
|           | LBP (8,1)  | 13656 | 74247 | 3531  | 140880 | 87918    | 0.665        | 0.155        | 0.345        |
|           | LBP (16,2) | 13212 | 53358 | 3975  | 161769 | 66588    | 0.753        | 0.198        | 0.248        |
|           | LBP (24,3) | 12918 | 35772 | 4269  | 179355 | 48693    | 0.827        | 0.265        | 0.166        |
| Döndürme  | Ö.Y.       | 5985  | 1659  | 12105 | 257811 | 7644     | <b>0.950</b> | <b>0.782</b> | <b>0.006</b> |
|           | LBP (8,1)  | 7947  | 85977 | 10152 | 173493 | 93927    | 0.653        | 0.084        | 0.331        |
|           | LBP (16,2) | 5673  | 59340 | 12420 | 200130 | 65028    | 0.741        | 0.087        | 0.228        |
|           | LBP (24,3) | 4422  | 38397 | 13671 | 221073 | 42819    | 0.812        | 0.103        | 0.147        |
| Ölçekleme | Ö.Y.       | 4044  | 1920  | 15027 | 247107 | 5964     | <b>0.936</b> | <b>0.678</b> | <b>0.007</b> |
|           | LBP (8,1)  | 7812  | 81861 | 11259 | 167166 | 89673    | 0.652        | 0.087        | 0.328        |
|           | LBP (16,2) | 6060  | 58368 | 13014 | 190659 | 64440    | 0.733        | 0.094        | 0.234        |
|           | LBP (24,3) | 4650  | 37119 | 14421 | 211908 | 41775    | 0.807        | 0.111        | 0.149        |
| Çarpıtma  | Ö.Y.       | 12021 | 1203  | 14499 | 251202 | 13224    | <b>0.943</b> | <b>0.909</b> | <b>0.004</b> |
|           | LBP (8,1)  | 11202 | 83145 | 15318 | 169260 | 94347    | 0.646        | 0.118        | 0.329        |
|           | LBP (16,2) | 8541  | 60015 | 17988 | 192390 | 68559    | 0.720        | 0.124        | 0.237        |
|           | LBP (24,3) | 6462  | 36279 | 20064 | 216126 | 42741    | 0.798        | 0.151        | 0.143        |
| Karma     | Ö.Y.       | 4455  | 2151  | 17358 | 255246 | 6606     | <b>0.930</b> | <b>0.674</b> | <b>0.008</b> |
|           | LBP (8,1)  | 8961  | 85623 | 12852 | 171774 | 94593    | 0.647        | 0.094        | 0.332        |
|           | LBP (16,2) | 6822  | 59820 | 14991 | 197577 | 66657    | 0.732        | 0.102        | 0.232        |
|           | LBP (24,3) | 4905  | 37668 | 16908 | 219729 | 42576    | 0.804        | 0.115        | 0.146        |

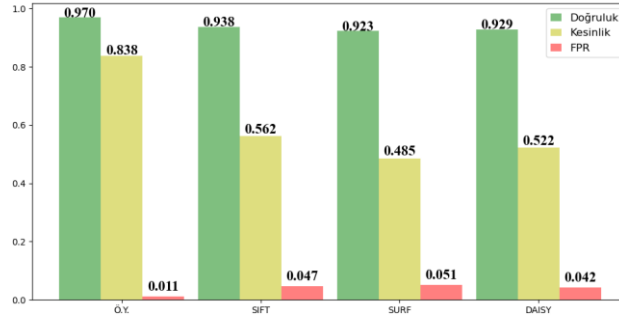
Tablo 5.8’de belirtilen karşılaştırma sonuçlarına göre, önerilen yöntemin COMOFOD veri setinde orijinal LBP algoritmalarından daha iyi sonuç verdiği gözlemlenmektedir.

Algoritmaya ait en etkili parametrelerin bulunması ve orijinal LBP algoritmaları ile karşılaştırılmasından sonra önerilen yöntem literatürdeki popüler algoritmalar olan SIFT, SURF ve DAISY algoritmaları ile kıyaslanmıştır. Tablo 5.9’da önerilen yöntem ile SIFT, SURF ve DAISY algoritmalarına ait COMOFOD veri seti kullanılarak ortaya çıkan eşleşme performansları gösterilmektedir.

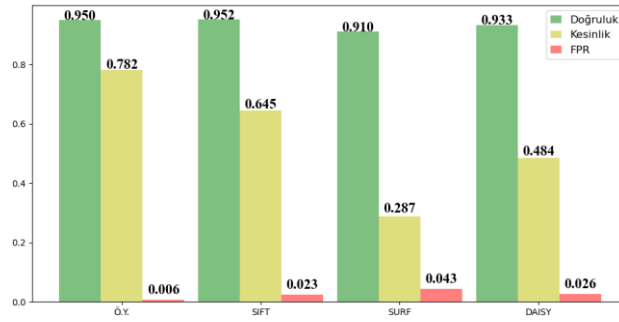
Tablo 5.9: Önerilen yöntem (Ö.Y.) ve SIFT, SURF, DAISY algoritmaları karşılaştırılması

| Atak      | Yöntem | TP    | FP    | FN    | TN     | KTS N.S. | Doğruluk     | Kesinlik     | FPR          |
|-----------|--------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Öteleme   | Ö.Y.   | 12885 | 2484  | 4302  | 212643 | 15369    | <b>0.970</b> | <b>0.838</b> | <b>0.011</b> |
|           | SIFT   | 13197 | 10260 | 3990  | 204867 | 23457    | 0.938        | 0.562        | 0.047        |
|           | SURF   | 10461 | 11076 | 6726  | 204051 | 21537    | 0.923        | 0.485        | 0.051        |
|           | DAISY  | 9966  | 9123  | 7221  | 206004 | 9966     | 0.929        | 0.522        | 0.042        |
| Döndürme  | Ö.Y.   | 5985  | 1659  | 12105 | 257811 | 7644     | 0.950        | <b>0.782</b> | <b>0.006</b> |
|           | SIFT   | 11148 | 6135  | 6942  | 253335 | 17283    | <b>0.952</b> | 0.645        | 0.023        |
|           | SURF   | 4521  | 11187 | 13569 | 248283 | 15711    | 0.910        | 0.287        | 0.043        |
|           | DAISY  | 6387  | 6798  | 11703 | 252672 | 13185    | 0.933        | 0.484        | 0.026        |
| Ölçekleme | Ö.Y.   | 4044  | 1920  | 15027 | 247107 | 5964     | <b>0.936</b> | <b>0.678</b> | <b>0.007</b> |
|           | SIFT   | 8895  | 8211  | 10176 | 240816 | 17106    | 0.931        | 0.519        | 0.032        |
|           | SURF   | 3234  | 11958 | 15837 | 237069 | 15198    | 0.896        | 0.212        | 0.048        |
|           | DAISY  | 6387  | 6798  | 11703 | 252672 | 13185    | 0.933        | 0.484        | 0.026        |
| Çarpıtma  | Ö.Y.   | 12021 | 1203  | 14499 | 251202 | 13224    | 0.943        | <b>0.909</b> | <b>0.004</b> |
|           | SIFT   | 17817 | 6354  | 8703  | 246051 | 24171    | <b>0.946</b> | 0.737        | 0.025        |
|           | SURF   | 7572  | 11463 | 18948 | 240942 | 19035    | 0.890        | 0.397        | 0.045        |
|           | DAISY  | 11535 | 5886  | 14985 | 246519 | 17421    | 0.925        | 0.662        | 0.023        |
| Karma     | Ö.Y.   | 4455  | 2151  | 17358 | 255246 | 6606     | 0.930        | <b>0.674</b> | <b>0.008</b> |
|           | SIFT   | 11277 | 8463  | 10536 | 248934 | 19740    | <b>0.931</b> | 0.571        | 0.032        |
|           | SURF   | 4254  | 12360 | 17559 | 245037 | 16614    | 0.892        | 0.256        | 0.048        |
|           | DAISY  | 6357  | 7644  | 15456 | 249753 | 14001    | 0.917        | 0.454        | 0.029        |

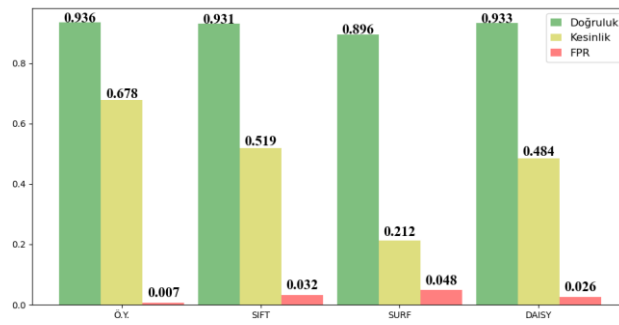
Tablo 5.9’da belirtilen COMOFOD veri seti üzerinden elde edilen karşılaştırma sonuçlarına göre, hazırlanan algoritma tüm ataklarda en yüksek hassasiyete ve en düşük yanlış pozitif algılama oranına sahiptir. Algoritmaların grafiksel karşılaştırmaları şekil 5.1-5.5’te gösterilmektedir.



Şekil 5.1: Önerilen yöntem, SIFT, SURF, DAISY öteleme atağı karşılaştırılması



Şekil 5.2: Önerilen yöntem, SIFT, SURF, DAISY döndürme atağı karşılaştırılması



Şekil 5.3: Önerilen yöntem, SIFT, SURF, DAISY ölçekleme atağı karşılaştırılması



Şekil 5.4: Önerilen yöntem, SIFT, SURF, DAISY çarpıtma atağı karşılaştırılması



Şekil 5.5: Önerilen yöntem, SIFT, SURF, DAISY karma atakların karşılaştırılması

Yukarıda gösterilen tablo ve şekillere göre tez algoritması genel olarak orijinal LBP algoritmaları, SIFT, SURF ve DAISY algoritmalarına göre daha etkin eşleme performansı göstermektedir. Bunun yanında CVIP veri setinden elde edilen sonuçlar Tablo 5.10'da gösterilmektedir. Şekil 5.6 ve 5.7'de tez algoritması ile yapılan kopyala taşı sahtecilik tespitine ait sonuçlar gösterilmektedir.

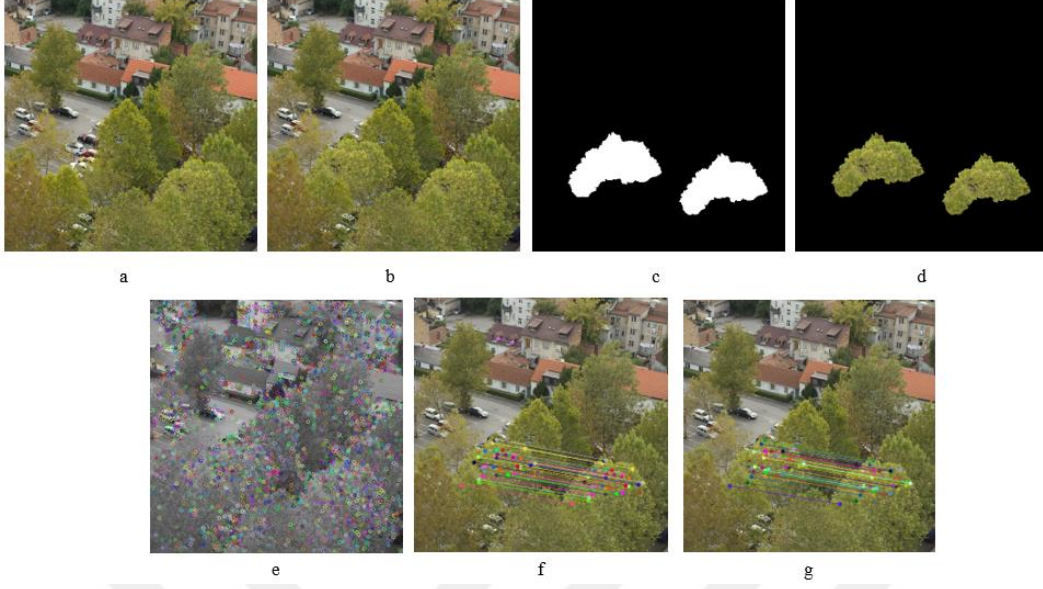
Tablo 5.10: CVIP veri seti üzerinde Önerilen yöntem (Ö.Y.) ve SIFT, SURF, DAISY algoritmaları karşılaştırılması

| Yöntem | Atak    | TP    | FP    | FN    | TN     | KTS N.S. | Doğruluk     | Kesinlik     | FPR          |
|--------|---------|-------|-------|-------|--------|----------|--------------|--------------|--------------|
| Ö.Y.   | Öteleme | 35187 | 1566  | 8505  | 461136 | 36753    | <b>0,980</b> | <b>0,957</b> | <b>0,003</b> |
| SIFT   |         | 35463 | 6432  | 8229  | 456270 | 41895    | 0,971        | 0,846        | 0,013        |
| SURF   |         | 27615 | 16803 | 16077 | 445899 | 44418    | 0,935        | 0,621        | 0,030        |
| DAISY  |         | 25908 | 6072  | 17784 | 456630 | 31980    | 0,952        | 0,810        | 0,013        |
| Ö.Y.   | R-15    | 654   | 384   | 9291  | 216015 | 1038     | 0,957        | <b>0,630</b> | <b>0,001</b> |
| SIFT   |         | 6033  | 2304  | 3912  | 214095 | 8337     | <b>0,972</b> | 0,723        | 0,010        |
| SURF   |         | 2634  | 8718  | 7311  | 207681 | 11352    | 0,929        | 0,230        | 0,040        |
| DAISY  |         | 2619  | 1737  | 7326  | 214662 | 4356     | 0,959        | 0,601        | 0,008        |
| Ö.Y.   | R-10    | 1755  | 402   | 8181  | 216081 | 2157     | 0,962        | <b>0,813</b> | <b>0,001</b> |
| SIFT   |         | 6096  | 2295  | 3840  | 214188 | 8391     | <b>0,972</b> | 0,726        | 0,010        |
| SURF   |         | 2880  | 8862  | 7056  | 207621 | 11742    | 0,929        | 0,245        | 0,040        |
| DAISY  |         | 3843  | 1740  | 6093  | 214743 | 5583     | 0,965        | 0,813        | 0,001        |
| Ö.Y.   | R-5     | 4434  | 411   | 5847  | 215934 | 4845     | 0,972        | <b>0,915</b> | <b>0,001</b> |
| SIFT   |         | 6492  | 2319  | 3789  | 214026 | 8811     | <b>0,973</b> | 0,736        | 0,010        |
| SURF   |         | 3384  | 8865  | 6897  | 207480 | 12249    | 0,930        | 0,276        | 0,040        |
| DAISY  |         | 4743  | 1743  | 5538  | 214602 | 6486     | 0,967        | 0,731        | 0,008        |
| Ö.Y.   | R-4     | 4707  | 417   | 5577  | 215967 | 5124     | <b>0,973</b> | <b>0,918</b> | <b>0,001</b> |
| SIFT   |         | 6459  | 2295  | 3825  | 214089 | 8754     | <b>0,973</b> | 0,737        | 0,010        |
| SURF   |         | 3327  | 8784  | 6957  | 207600 | 12111    | 0,939        | 0,274        | 0,405        |
| DAISY  |         | 4692  | 1755  | 5592  | 214629 | 6447     | 0,967        | 0,727        | 0,008        |
| Ö.Y.   | R-3     | 5001  | 414   | 5238  | 215973 | 5415     | <b>0,975</b> | <b>0,923</b> | <b>0,001</b> |
| SIFT   |         | 6408  | 2307  | 3831  | 214080 | 8715     | 0,972        | 0,735        | 0,010        |
| SURF   |         | 3225  | 8826  | 7014  | 207561 | 12051    | 0,930        | 0,026        | 0,040        |
| DAISY  |         | 4713  | 1764  | 5526  | 214623 | 6477     | 0,967        | 0,727        | 0,001        |
| Ö.Y.   | R-2     | 5268  | 408   | 4914  | 215937 | 5676     | <b>0,976</b> | <b>0,928</b> | <b>0,001</b> |
| SIFT   |         | 6417  | 2307  | 3765  | 214038 | 8724     | 0,973        | 0,735        | 0,010        |
| SURF   |         | 3249  | 8814  | 6933  | 207531 | 12063    | 0,930        | 0,269        | 0,040        |
| DAISY  |         | 4872  | 1743  | 5310  | 214602 | 6615     | 0,968        | 0,736        | 0,008        |
| Ö.Y.   | R-1     | 5292  | 420   | 4854  | 215817 | 5712     | <b>0,976</b> | <b>0,926</b> | <b>0,008</b> |
| SIFT   |         | 6453  | 2316  | 3693  | 213921 | 8769     | 0,973        | 0,737        | 0,017        |
| SURF   |         | 3303  | 8875  | 6843  | 207462 | 12078    | 0,931        | 0,273        | 0,040        |
| DAISY  |         | 4827  | 1731  | 5319  | 214506 | 6558     | 0,968        | 0,736        | 0,008        |
| Ö.Y.   | R0      | 7506  | 399   | 2349  | 215868 | 7905     | <b>0,987</b> | <b>0,949</b> | <b>0,001</b> |
| SIFT   |         | 7644  | 2298  | 2211  | 213969 | 9942     | 0,980        | 0,768        | 0,010        |
| SURF   |         | 6375  | 8793  | 3480  | 207474 | 15168    | 0,945        | 0,420        | 0,040        |
| DAISY  |         | 5967  | 1752  | 3888  | 214515 | 7719     | 0,975        | 0,773        | 0,008        |
| Ö.Y.   | R1      | 5250  | 405   | 4917  | 215817 | 5655     | <b>0,976</b> | <b>0,928</b> | <b>0,001</b> |
| SIFT   |         | 6303  | 2310  | 3864  | 213912 | 8613     | 0,972        | 0,731        | 0,010        |
| SURF   |         | 3261  | 8787  | 6906  | 207435 | 12048    | 0,930        | 0,270        | 0,040        |
| DAISY  |         | 4785  | 1755  | 5382  | 214467 | 6540     | 0,968        | 0,731        | 0,008        |
| Ö.Y.   | R2      | 5169  | 408   | 5094  | 215937 | 5577     | <b>0,975</b> | <b>0,926</b> | <b>0,001</b> |
| SIFT   |         | 6390  | 2289  | 3873  | 214056 | 8679     | 0,972        | 0,736        | 0,010        |
| SURF   |         | 3399  | 8814  | 6864  | 207531 | 12213    | 0,930        | 0,278        | 0,040        |
| DAISY  |         | 4863  | 1752  | 5400  | 214593 | 6615     | 0,968        | 0,735        | 0,008        |
| Ö.Y.   | R3      | 5031  | 426   | 5244  | 215841 | 5457     | <b>0,974</b> | <b>0,921</b> | <b>0,001</b> |
| SIFT   |         | 6411  | 2295  | 3864  | 213972 | 8706     | 0,972        | 0,736        | 0,010        |
| SURF   |         | 3246  | 8811  | 7029  | 207456 | 12057    | 0,930        | 0,269        | 0,040        |
| DAISY  |         | 4611  | 1758  | 5664  | 214509 | 6369     | 0,967        | 0,723        | 0,008        |
| Ö.Y.   | R4      | 4797  | 414   | 5463  | 215898 | 6462     | <b>0,974</b> | <b>0,920</b> | <b>0,001</b> |
| SIFT   |         | 6393  | 2319  | 3867  | 213993 | 8712     | 0,972        | 0,733        | 0,010        |

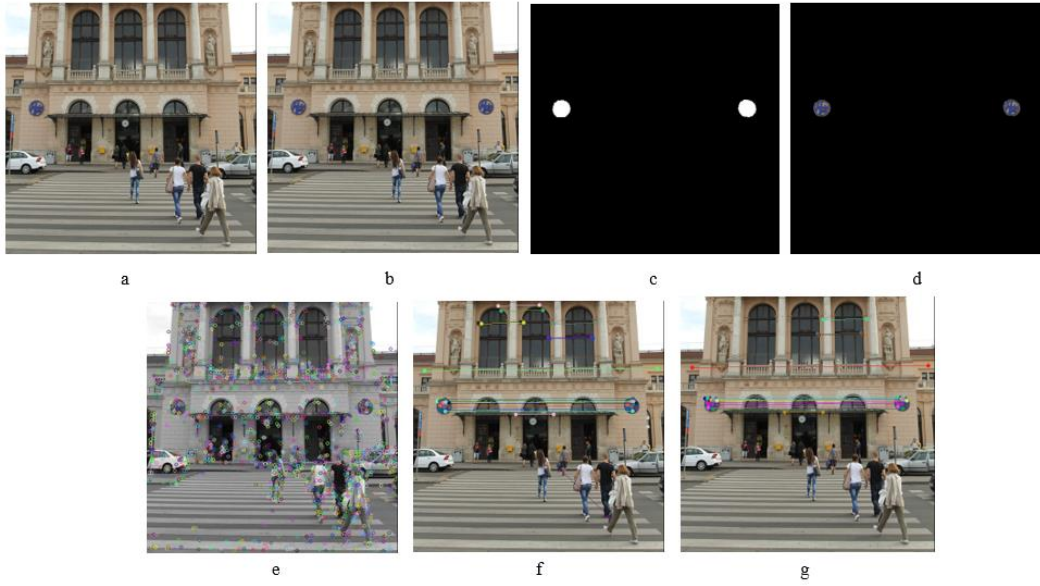
|       |           |      |      |       |        |       |              |              |              |
|-------|-----------|------|------|-------|--------|-------|--------------|--------------|--------------|
| SURF  |           | 3363 | 8820 | 6897  | 207492 | 12183 | 0,930        | 0,276        | 0,040        |
| DAISY |           | 4692 | 1770 | 5568  | 214542 | 6462  | 0,967        | 0,726        | 0,008        |
| Ö.Y.  | R5        | 4365 | 408  | 5820  | 215937 | 4773  | 0,972        | 0,914        | 0,001        |
| SIFT  |           | 6318 | 2298 | 3867  | 214047 | 8616  | 0,972        | 0,733        | 0,010        |
| SURF  |           | 3324 | 8919 | 6861  | 207426 | 12243 | 0,930        | 0,271        | 0,041        |
| DAISY |           | 4653 | 1776 | 5532  | 214596 | 6429  | 0,967        | 0,723        | 0,008        |
| Ö.Y.  |           |      |      |       |        |       |              |              |              |
| Ö.Y.  | R10       | 1851 | 414  | 8046  | 216000 | 2265  | 0,962        | <b>0,817</b> | <b>0,001</b> |
| SIFT  |           | 6114 | 2304 | 3783  | 214110 | 8418  | <b>0,973</b> | 0,726        | 0,010        |
| SURF  |           | 2889 | 8886 | 7008  | 207528 | 11775 | 0,929        | 0,245        | 0,004        |
| DAISY |           | 3726 | 1746 | 6171  | 214668 | 5472  | 0,965        | 0,680        | 0,008        |
| Ö.Y.  | R15       | 741  | 417  | 9228  | 215937 | 1158  | 0,957        | 0,639        | <b>0,001</b> |
| SIFT  |           | 5976 | 2268 | 3993  | 214086 | 8244  | <b>0,972</b> | <b>0,724</b> | 0,010        |
| SURF  |           | 2724 | 8901 | 7245  | 207453 | 11625 | 0,928        | 0,234        | 0,041        |
| DAISY |           | 2490 | 1731 | 7479  | 214623 | 4221  | 0,959        | 0,589        | 0,008        |
| Ö.Y.  | S<br>0.75 | 540  | 420  | 7383  | 217434 | 960   | 0,965        | 0,562        | <b>0,001</b> |
| SIFT  |           | 3462 | 2295 | 4461  | 215559 | 5757  | <b>0,970</b> | <b>0,601</b> | 0,010        |
| SURF  |           | 960  | 9105 | 6963  | 208749 | 10065 | 0,928        | 0,095        | 0,041        |
| DAISY |           | 1542 | 1734 | 6381  | 216120 | 3276  | 0,964        | 0,470        | 0,007        |
| Ö.Y.  | S0.8      | 990  | 423  | 7215  | 217194 | 1413  | 0,966        | <b>0,700</b> | <b>0,001</b> |
| SIFT  |           | 4035 | 2295 | 4170  | 215322 | 6330  | <b>0,971</b> | 0,637        | 0,010        |
| SURF  |           | 1110 | 9102 | 7095  | 208515 | 10212 | 0,928        | 0,108        | 0,041        |
| DAISY |           | 2379 | 1731 | 5826  | 215886 | 4110  | 0,966        | 0,578        | 0,007        |
| Ö.Y.  | S<br>0.85 | 1866 | 420  | 6750  | 216906 | 2286  | 0,968        | <b>0,816</b> | <b>0,001</b> |
| SIFT  |           | 4347 | 2292 | 4269  | 215034 | 6639  | <b>0,970</b> | 0,654        | 0,010        |
| SURF  |           | 1296 | 9021 | 7320  | 208305 | 10317 | 0,927        | 0,125        | 0,041        |
| DAISY |           | 2994 | 1755 | 5622  | 215571 | 4749  | 0,967        | 0,630        | 0,008        |
| Ö.Y.  | S<br>0.90 | 3015 | 402  | 6000  | 216534 | 3417  | <b>0,971</b> | <b>0,882</b> | <b>0,001</b> |
| SIFT  |           | 4875 | 2271 | 4140  | 214665 | 7146  | <b>0,971</b> | 0,682        | 0,010        |
| SURF  |           | 1647 | 8991 | 7368  | 207945 | 10638 | 0,927        | 0,154        | 0,041        |
| DAISY |           | 3516 | 1725 | 5499  | 215211 | 5241  | 0,968        | 0,670        | 0,007        |
| Ö.Y.  | S<br>0.95 | 4332 | 429  | 5046  | 216243 | 4761  | <b>0,975</b> | <b>0,909</b> | <b>0,001</b> |
| SIFT  |           | 5511 | 2283 | 3867  | 214389 | 7794  | 0,972        | 0,707        | 0,010        |
| SURF  |           | 2514 | 8883 | 6864  | 207789 | 11397 | 0,930        | 0,220        | 0,040        |
| DAISY |           | 4032 | 1737 | 5346  | 214935 | 5769  | 0,968        | 0,698        | 0,008        |
| Ö.Y.  | S<br>1.05 | 5019 | 405  | 5487  | 215592 | 5424  | <b>0,973</b> | <b>0,925</b> | <b>0,001</b> |
| SIFT  |           | 5967 | 2277 | 4539  | 213720 | 8244  | 0,969        | 0,723        | 0,010        |
| SURF  |           | 2766 | 8952 | 7740  | 207045 | 11718 | 0,926        | 0,236        | 0,041        |
| DAISY |           | 5019 | 405  | 5487  | 215592 | 5424  | 0,973        | 0,925        | 0,001        |
| Ö.Y.  | S1.1      | 4182 | 405  | 7203  | 215331 | 4587  | <b>0,966</b> | <b>0,911</b> | <b>0,001</b> |
| SIFT  |           | 6054 | 2301 | 5331  | 213435 | 8355  | <b>0,966</b> | 0,724        | 0,010        |
| SURF  |           | 2160 | 8880 | 9225  | 206856 | 11040 | 0,920        | 0,195        | 0,041        |
| DAISY |           | 4341 | 1743 | 7044  | 213993 | 6084  | 0,961        | 0,713        | 0,008        |
| Ö.Y.  | S<br>1.15 | 3222 | 423  | 8670  | 214920 | 3645  | 0,959        | <b>0,883</b> | <b>0,001</b> |
| SIFT  |           | 6048 | 2283 | 5844  | 213060 | 8331  | <b>0,964</b> | 0,725        | 0,010        |
| SURF  |           | 2079 | 8940 | 9813  | 206403 | 11019 | 0,917        | 0,188        | 0,041        |
| DAISY |           | 4218 | 1737 | 7674  | 213606 | 5955  | 0,958        | 0,708        | 0,008        |
| Ö.Y.  | S1.2      | 2232 | 432  | 10260 | 214518 | 2664  | 0,952        | <b>0,837</b> | <b>0,002</b> |
| SIFT  |           | 6009 | 2343 | 6483  | 212607 | 8352  | <b>0,961</b> | 0,719        | 0,010        |
| SURF  |           | 1827 | 8814 | 10665 | 206136 | 10641 | 0,914        | 0,171        | 0,041        |
| DAISY |           | 3792 | 1758 | 9700  | 213192 | 5550  | 0,954        | 0,683        | 0,008        |
| Ö.Y.  | S<br>1.25 | 1623 | 423  | 11667 | 214029 | 2046  | 0,946        | <b>0,793</b> | <b>0,001</b> |
| SIFT  |           | 6111 | 2331 | 7179  | 212121 | 6442  | <b>0,958</b> | 0,723        | 0,010        |
| SURF  |           | 2082 | 8823 | 11208 | 205629 | 10905 | 0,912        | 0,190        | 0,041        |
| DAISY |           | 3369 | 1761 | 9921  | 212691 | 5130  | 0,948        | 0,656        | 0,008        |

CVIP veri setinden elde edilen sonuçlar Tablo 5.10'da gösterilmektedir. Tabloda ataklar kolonunda belirtilen R değeri döndürmeyi, S değeri ise ölçeklendirme atağını ifade etmektedir. Bunun yanında R ve S harflerinin yanında bulunan rakamlar ise yapılan atağın derecesini göstermektedir.

Kopyala taşı sahteciliği tespiti kapsamında hazırlanan bu tez çalışmasında ilk olarak tez çalışmasına ait en uygun parametrelerin tespiti yapılmıştır. Bu işlem için COMOFOD veri seti kullanılmıştır. En iyi parametrelerin tespitinden sonra ise çalışma, SIFT, SURF, DAISY ve üç farklı LBP algoritması ile kıyaslanmış ve başarımlar ölçümleri yukarıdaki tablolarda gösterilmiştir. Bunun yanında tez çalışmasının tespit performansı farklı bir veri seti olan CVIP veri seti üzerinde test edilmiş ve sonuçlar Tablo 5.10'da gösterilmiştir. Elde edilen sonuçlar ışığında önerilen yöntem COMOFOD veri setinde tüm ataklarda en yüksek hassasiyete ve en düşük yanlış pozitif algılama oranına sahiptir. Bunun nedeni önerilen yöntemin diğer algoritmalarından daha az yanlış nokta tespit etmesidir. Önerilen yöntemin daha az yanlış nokta tespit etmesi sonucu, daha düşük FPR ve daha yüksek keskinlik oranı ortaya çıkarmaktadır. Buna ek olarak, önerilen yöntem öteleme ve ölçekleme ataklarında en iyi doğruluk oranına sahiptir, ancak diğer işlemler için çok az farkla SIFT algoritmasının gerisinde yer almaktadır. Bunun yanında önerilen yöntem CVIP veri setinde öteleme atağında en iyi performansa sahiptir. Aynı veri seti üzerinde döndürme atağında -5,5 değer aralıklarında, ölçekleme atağında 0.9, 1.1 aralığında en iyi doğruluk, kesinlik ve yanlış tahmin oranı değerine sahiptir. Diğer aralıklarda ise az farkla SIFT algoritmasının gerisinde kalmaktadır. Bu tez çalışmasındaki amaç hazırlanan algoritmanın kopyala taşı sahtecilik alanında uygun ve kullanılabilir bir algoritma olduğunu göstermektir. Şekil 5.6'da kopyala taşı sahteciliği yapılmış görüntülere ait sonuçlar hiyerarşik olarak gösterilmektedir.



Şekil 5.6: COMOFOFOD-13 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Önerilen yöntem eşleme sonrası g) RANSAC sonrası



Şekil 5.7: COMOFOFOD-70 a) Orijinal görüntü b) Bozulmuş görüntü c) Bozulmuş görüntü ikili maskesi d) Bozulmuş görüntü renkli maske e) SIFT köşe noktaları f) Önerilen yöntem eşleme sonrası g) RANSAC sonrası

## 6.SONUÇ VE ÖNERİLER

Bu bölümde tez çalışması kapsamında elde edilen bulgulara ait sonuçlar tartışılacaktır. Tez çalışması kapsamında ilk olarak hazırlanan yöntemin en optimum parametreleri tespit edilmiştir. COMOFOD veri seti üzerinde elde edilen sonuçlara dayanarak 15x15 dörtgensel alan, 2 iterasyonlu Manhattan uzaklığı parametreleri kullanılarak elde edilen sonuçların en etkili eşleşme performansını verdiği gözlemlenmiştir. Bunun yanında LBP tabanlı olan önerilen yöntem diğer LBP tabanlı yöntemler ile kıyaslanmıştır. Sonuç olarak tüm ataklarda diğer LBP algoritmalarına karşı daha başarılı sonuçlar verdiği gözlemlenmiştir.

Optimum parametreler ve temel LBP algoritmaları ile performans karşılaştırılmasından sonra önerilen yöntem, SIFT, SURF ve DAISY algoritmaları ile karşılaştırılmıştır. Karşılaştırma işlemi için COMOFOD ve CVIP veri setleri kullanılmıştır.

COMOFOD veri seti üzerinden elde edilen karşılaştırma sonuçlarına göre, hazırlanan algoritma tüm ataklarda en yüksek hassasiyete ve en düşük yanlış pozitif algılama oranına sahiptir. Bunun nedeni önerilen yöntemin diğer algoritmalarından daha az yanlış nokta tespit etmesidir. Önerilen yöntemin daha az yanlış nokta tespit etmesi sonucu, daha düşük FPR ve daha yüksek keskinlik oranı ortaya çıkarmaktadır. Buna ek olarak, önerilen yöntem öteleme ve ölçekleme ataklarında en iyi doğruluk oranına sahiptir, ancak diğer işlemler için çok az farkla ikinci sıradadır. Kopyala taşı sahtecilik kapsamında hazırlanan yöntem elde edilen sonuçlara göre öteleme atağına karşı tüm rakiplerinden daha etkili sonuçlar vermektedir.

CVIP veri setinden elde edilen sonuçlara göre, hazırlanan algoritma tüm karşılaştırma alanlarında en yüksek kesinlik ve en düşük FPR oranına sahiptir. Önerilen yöntem -15,15 derece aralığında yapılan rotasyon saldırılarında -5,5 derece aralığında en yüksek doğruluk değerine sahiptir. Bunun yanında 0.75,1.25 aralığı baz alınarak yapılan ölçekleme ataklarında 0.9,1.1 aralığında en yüksek doğruluk oranına sahiptir. En iyi sonuçların tespit edildiği aralıklar dışında kalan aralıklarda ise az farkla SIFT algoritmasından geride ikinci sırada yer almaktadır.

Kopyala taşı sahteciliği kapsamında hazırlanan yönteme ait sonuçlar incelendiği zaman, hazırlanan algoritma tüm karşılaştırmalarda en düşük FPR değerine ve en yüksek kesinlik değerine sahiptir. İki veri setinde de öteleme atağında tüm rakiplerinden daha yüksek eşleşme performansı göstermiştir. Ayrıca belirli parametreler içerisinde en yüksek doğruluk oranına sahip algoritma diğer aralıklarda ise az farkla ikinci sıradadır. Önerilen yöntemin katı eşleşme yapısı sonuç olarak tüm karşılaştırmalarda düşük FPR oranı ortaya çıkarmıştır. Fakat bu yapı aynı zamanda pozitif eşleşme sayısını da etkilediği düşünülmektedir. Olumlu sonuçlar elde edilmesine rağmen daha fazla pozitif eşleşme sayısı elde edilebilmesi için algoritma geliştirilmelidir. Bunun için ilerleyen aşamalarda algoritmanın katı yapısı esnetilebilir. Bunun yanında önerilen yöntemin ölçekleme performansını geliştirebilmek için ilerleyen aşamalarda çok katmanlı yaklaşım entegre edilebilir. Günümüzde gelişen yapay zekâ teknolojileri incelenerek öznitelik oluşturma veya eşleştirme aşamalarında bu teknolojilerin entegre edilmesi çalışmanın özgünlüğüne katkı sağlayacağı düşünülmektedir.

## KAYNAKLAR DİZİNİ

- Agarwal, R., Prakash, O. V.,** 2019, An efficient copy move forgery detection using deep learning feature extraction and matching algorithm, *Multimedia Tools and Applications*, 79:7355-7376
- AlSawadi, M., Muhammad, G., Hussain, M. and Bebis, G.,** 2013, Copy Move Image Forgery Detection Using Local Binary Pattern and Neighborhood Clustering, *European Modelling Symposium*
- Ardizzone E., Mazzola G.,** 2015, A Tool To Support The Creation of Datasets of Tampered Videos, *I.C on Image Analysis and Processing, ICIAP 2015*
- Ashwini, P.,** 2014, Image Mosaicing using SIFT and Corner Detection Algorithm, *International Journal of Advanced Technology and Engineering Research (IJATER)* ,4, (2), pp. 51-57
- Badal, S., Pradip. K., et al.,** 2018, Keypoints based enhanced multiple copymove forgeries detection system using density-based spatial clustering of application with noise clustering algorithm, *IET Image Processing*, 12, (11), pp. 2092-2099
- Bayram, S., Sencar, H.T., et al.,** 2009, An efficient and robust method for detecting copy-move forgery, *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Taipei, Taiwan, pp. 1053–1056
- Bo, X., Junwen, W., Guangjie, L. and Yuewei, D.,** 2010, ‘Image copy-move forgery detection based on SURF’, *Int. Conf. on Multimedia Information Networking and Security*, Jiangsu, China, pp. 889–892
- Boz, A., Bilge H.S,** 2016, Copy-move image forgery detection based on LBP and DCT, *Signal Processing and Communication Application Conference (SIU)*, pp. 561-564
- Bravo-Solorio, S. and Nandi, A.K.,** 2020, Exposing duplicated regions affected by reflection, rotation and scaling, *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 1880–1883 *IET Image Process*, pp. 1–8 c *The Institution of Engineering and Technology*
- Brown, M. and Lowe D. G.,** 2003, Recognizing Panoramas, *IEEE Int. Conf. on Computer Vision*

### KAYNAKLAR DİZİNİ (devam)

- Chia, W., Yeong, L., et al.,** 2014, The Effect of Rainfall on Feature Points Extraction and Image Stitching. Int. Conf. on Information Science, Electronics and Electrical Engineering,
- Dağdeviren F.,** 2015, Sürü Zekası Yaklaşımı ile Panoramik Görüntü Oluşturma,
- Davarzani, R., Yaghmaie, S., et al.,** 2013, Copy-Move Forgery Detection using Multiresolution Local Binary Patterns, Forensic Science International , 2013,231, (1-3), pp. 61-72
- Fridrich, J., Soukalm, D. and Lukas, J.,** 2003, Detection of copy-move forgery in digital images, Proc. Digital Forensic Research Workshop
- Fischler, M.A. and Bolles, R.,** 1981, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, in Proc. Image Understanding Workshop, pp. 71-88
- Guo, Z., Zhang, L. and Zhang, D.,** 2010, A Completed Modelling of Local Binary Patterns for Texture Classification, IEEE Transactions on Image Processings, 19, (6), pp. 1657-1663
- Hemlata, J.,** 2013, Image Mosaicing using Harris, SIFT Feature Detection Algorithm, Internation Journal of Science,Engineering and Technology research (IJSETR), ISSN:2278-7798
- Huang, Y., Lu, W., Sun, W. and Long, D.,** 2011, Improved DCT-based detection of copy move forgery in images, Forensic Science International, ,206, (1-3), pp. 178-184
- Huang. D., Shan, C., et al.,** 2011, Local Binary Patterns and Its Application to Facial Image Analysis: A Survey, International Journal of Engineering Development and Research , ,41, (46), pp. 765-781
- Kaur, R. and Kaur,** 2016, Copy-Move Forgery Detection using ORB and SIFT Detector, International Journal of Engineering Development and Research ,4, (4), pp. 804-813

### KAYNAKLAR DİZİNİ (devam)

- Kuznetsov, A.**, 2019, Digital image forgery detection using deep learning approach, Journal of Physics: Conference Series
- Li li**, 2014, Image Matching Algorithm based on DAISY Descriptor, Journal of Multimedia
- Muzaffer, G., Ulutaş, G., et al.**, 2018, SIFT ve SURF Tabanlı Sayısal Görüntü Sahteciliği Tespiti Yöntemi. 26th Signal Processing and Communications Applications Conference (SIU)
- M. Yu, X. Gu and Wang, Y.**, 2010, Histogram similarity using variable bin size distance, Elsevier Computer Vision And Image Understanding,114, (8), pp. 981-989
- Ojala, T., M. Pietikainen, et al.**, 2002, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, IEEE Transactions on Pattern Analysis and Machine Intelligence,34, (7), pp. 971-987
- Peng, K. and Hongbing, M.**, 2011, An Automatic Image Mosaicing Method Based on SIFT Feature Matching, IEEE 978-1-61284-774-0/11
- Prasad, S., Ramkumar, B.**, 2016, Copy Move Forgery Detection using SIFT, HOG and SURF Features, Int. Conf. On Recent Trends In Electronics Information Communication Technology, Bangalore, India, pp. 706–710
- Popescu, A.C. and Farid, H.**, 2004, Exposing digital forgeries by detecting duplicated image regions, Tech. Rep. TR2004-515, Dartmouth College
- Savran, İ., Mumcu, A.**, 2018, Detection of Copy Move Forgery Technique Based on SIFT and SURF. 26th Signal Processing and Communications Applications Conference (SIU)
- Sci-kit Documents**, 2016, [http://scikit-image.org/docs/dev/auto\\_examples/features\\_detection/plot\\_daisy.html](http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_daisy.html)
- Teerakanok, S. and Uehara, T.**, 2019, Copy-move forgery detection : a State-of-the-Art Technical Review and Analysis, IEEE Access,,7, pp. 40550-40568

### KAYNAKLAR DİZİNİ (devam)

- Lowe, D., G.,** 2004, Distinctive image features from Scale-Invariant key points, International Journal of Computer Vision,60, pp. 91-110
- Tralic D., Zupancic, L., et al.,** 2012, COMOFOOD – New Database for Copy-Move Forgery Detection, in Proc. 55th International Symposium
- Velardo, C. and Dugelay, J.,** 2010, Face Recognition with DAISY Descriptors, ACM
- Waleed, J. and Abdullah, D.A.,** 2018, Comprehensive Display of Digital Image Copy-Move Forensics Techniques, Int. Conf. on Engineering Technologies and their Applications (ICETA), ALNajaf, IRAQ., pp. 155-160
- Fischler, M.A. and Bolles, R.,** 1981, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, in Proc. Image Understanding Workshop, pp. 71-88
- Tralic D., Zupancic, L., et al.,** 2012, COMOFOOD – New Database for Copy-Move Forgery Detection, in Proc. 55th International Symposium
- Lowe, D., G.,** 2004, Distinctive image features from Scale-Invariant key points, International Journal of Computer Vision,60, pp. 91-110
- Huang. D., Shan, C., et al.,** 2011, Local Binary Patterns and Its Application to Facial Image Analysis: A Survey, International Journal of Engineering Development and Research , ,41, (46), pp. 765-781
- Badal, S., Pradip. K., et al.,** 2018, Keypoints based enhanced multiple copymove forgeries detection system using density-based spatial clustering of application with noise clustering algorithm, IET Image Processing, 12, (11), pp. 2092-2099
- Dağdeviren F.,** 2015, Sürü Zekası Yaklaşımı ile Panoramik Görüntü Oluşturma,
- Hemlata, J.,** 2013, Image Mosaicing using Harris, SIFT Feature Detection Algorithm, Internation Journal of Science,Engineering and Technology research (IJSETR), ISSN:2278-7798
- Peng, K. and Hongbing, M.,** 2011, An Automatic Image Mosaicing Method Based on SIFT Feature Matching, IEEE 978-1-61284-774-0/11
- Sci-kit Documents,** 2016, [http://scikit-image.org/docs/dev/auto\\_examples/features\\_detection/plot\\_daisy.html](http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_daisy.html)

### KAYNAKLAR DİZİNİ (devam)

- Teerakanok, S. and Uehara, T.,** 2019, Copy-move forgery detection : a State-of-the-Art Technical Review and Analysis, IEEE Access,,7, pp. 40550-40568
- Tralic D., Zupancic, L., et al.,** 2012, COMOFOFOD – New Database for Copy-Move Forgery Detection, in Proc. 55th International Symposium
- Velardo, C. and Dugelay, J.,** 2010, Face Recognition with DAISY Descriptors, ACM
- Waleed, J. and Abdullah, D.A.,** 2018, Comprehensive Display of Digital Image Copy-Move Forensics Techniques, Int. Conf. on Engineering Technologies and their Applications (ICETA), ALNajaf, IRAQ,, pp. 155-160
- Wang G., Rister B. and Cavallaro J.,** 2013, Workload Analysis and Efficient OpenCL-based Implementation of SIFT Algorithm on a Smartphone, IEE Global Conf. on Signal and Information Processing
- X. Wu and J. Sun.,** 2017, Joint-scale LBP: a new feature descriptor for texture classification, The Visual Computer,33, pp. 317-329
- Zhao, J. and Zhao, W.,** 2013, Passive Forensics for Region Duplication Image Forgery Based on Harris Feaure Points and Local Binary Patterns, Mathematical Problems in Engineering , pp. 1-12



## TEŞEKKÜR

Doktora eğitimim süresince ve tez çalışmamda benden desteğini ve ilgisini esirgemeyen, bilgi ve deneyimleriyle yol gösteren tez danışmanım Sayın Prof. Dr. Serdar KORUKOĞLU ve Doç. Dr. Muhammed Gökhan CİNSDİKİCİ'ye sabrından, özverisinden ve cesaretlendirmesinden dolayı çok teşekkür ederim. Onların değerli fikirleri ve yardımları sayesinde bu tez ortaya çıkmıştır. Ayrıca hayatımın her döneminde benden desteğini esirgemeyen hayat arkadaşım eşime, anneme, babama ve kardeşlerime, doktora eğitimim süresince bana destek olan UNIPA A.Ş.'deki çalışma arkadaşlarıma sonsuz teşekkürlerimi sunarım.

.... / .... / 20..

İmzası

Fatih DAĞDEVİREN



## FATİH DAĞDEVİREN

## İ Ş D E N E Y İ M İ

❖ **Yazılım ve Arge Müdürü****11/2019-Devam**

ÜNİPA / İZMİR

Proje Yürütücüsü olarak şirket bünyesindeki projelere ait yazılım tasarlanması, geliştirilmesi ve geçerli kalmasından sorumluyum.

❖ **Kıdemli Yazılım Uzmanı****05/2018-11/2019**

ASİSGUARD / İZMİR

Görüntüleme Sistemleri, Görüntü işleme, Gömülü yazılım, Web ve masaüstü yazılımları, Arge faaliyetleri

❖ **Kıdemli Yazılım Uzmanı****07/2015-05/2018**

ÜNİPA / İZMİR

Kıdemli Yazılım Uzmanı olarak şirket bünyesinde hizmet verilen Öğrenci Bilgi Sistemi ve Bilimsel Araştırma Projeleri bilgi giriş sisteminde yazılım geliştirme rolünde görev aldım.

❖ **Yazılım Takım Lideri****02/2014-03/2015**

ÜNİPA / İZMİR

Yazılım takım lideri olarak şirket bünyesinde hizmet verilen Öğrenci Bilgi Sistemi ve Bilimsel Araştırma Projeleri bilgi giriş sistemi yönetiminde görev aldım.

❖ **Yazılım Mühendisi****01/2013-02/2014**

ÜNİPA / İZMİR

Junior Yazılım Geliştiricisi olarak Bilimsel Araştırma Projeleri bilgi girişi sisteminde görev aldım.

❖ **Doktora****09/2015-Devam**

EGE ÜNİVERSİTESİ / İZMİR

Ege Üniversitesi Fen Bilimleri Enstitüsü Uluslararası Bilgisayar / Bilgi Teknolojileri,  
Doktora, Tez Aşaması,

Tez Konusu: Görüntülerde Kopyala Taşı Sahtecilik Tespiti.

Kopyala Taşı Sahtecilik Tespiti görüntülerde gizlenmek istenen alanların aynı görüntü içerisindeki başka bir alan ile kapatılması ile yapılan sahtecilik yöntemidir. Tez kapsamında literatürdeki eşleme algoritmaları ile yarışacak yeni bir algoritma tasarımı yapılacaktır. Daha sonra tasarlanan algoritma ve literatürdeki diğer algoritmaların sahtecilik tespiti başarısı tartışmaya sunulacaktır. Offline bir sistemdir. Daha çok kriminal vakalarda ortaya çıkmaktadır.

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- QT
- Görüntü İşleme Algoritmaları
- Opencv
- Python 3
- C Programlama Dili

❖ **Yüksek Lisans****09/2012-06/2015**

EGE ÜNİVERSİTESİ / İZMİR

Ege Üniversitesi Fen Bilimleri Enstitüsü Uluslararası Bilgisayar / Bilgi  
Teknolojileri, Tezli Yüksek Lisans,

Tez Konusu: Sürü Zekâsı kullanarak Panoramik Görüntü oluşturma

*Tez çalışmasında yapay zekâ algoritmalarından sürü zekâsı algoritması ile Genetik algoritmanın birleşmesi sonucu elde edilen yeni bir algoritma ile görüntülerde tespit edilen köşe noktaları eşleştirilmiştir. Eşleşmeden sonra görüntüler üst üste çakıştırılarak birleştirilmektedir. Bu şekilde Panoramik görüntü elde edilmiştir.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- C#
- Yapay Arı Kolonisi Algoritması
- Genetik Algoritma
- Aforge Kütüphanesi

❖ **Lisans****09/2007-06/2012**

İZMİR EKONOMİ ÜNİVERSİTESİ / İZMİR

İzmir Ekonomi Üniversitesi Bilgisayar Bilimleri Fakültesi / Bilgisayar Mühendisliği,  
Lisans

*Lisans Bitirme Projesi: Doktor – Hasta iletişimi üzerine kurulmuş çeşitli görüntü işleme yöntemleri ile desteklenmiş Web uygulaması*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- C#
- Aforge Kütüphanesi
- WebForms
- JavaScript

## P R O J E L E R

### ❖ Öğrenci Bilgi Sistemi

**UNİPA / İZMİR**

*Açıklama: Öğrenci, Öğretim Üyeleri ve Öğrenci işleri tarafından yürütülen Üniversite bazlı bilgi sistemi. Projede yazılım geliştirme, analiz ve test süreçlerinde yer aldım.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Nhibernate
- Entity Framework
- Dapper
- MS-SQL
- JSon
- MVC
- WebForm
- Javascript
- JQuery
- Bootstrap
- C#
- SOAP
- REST Servisler
- Xml

### ❖ Bilimsel Araştırma Projeleri Bilgi Sistemi

**UNİPA / İZMİR**

*Açıklama: Öğretim Üyelerinin Üniversite içi/dışı proje bilgi girişlerinin gerçekleştirildiği sistemdir. Projede yazılım geliştirme, analiz ve test süreçlerinde yer aldım.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Nhibernate
- MS-SQL
- WebForm
- Javascript
- JQuery
- C#

### ❖ Karakutu Projesi

**ASİSGUARD / İZMİR**

*Açıklama: Araçların CAN haberleşme hattı üzerinden gelen verileri kaydetmek için hazırlanan api. Projede api tasarım aşamasında görev aldım.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Entity Framework
- MS-SQL
- JSon

- C#
- CAN Haberleşme Hattı
- PostgreSQL

#### ❖ Sürücü Bilgi Sistemi

#### ASİSGUARD / İZMİR

*Açıklama: CAN Haberleşme hattı üzerinden gelen verilerin işlenip zırhlı araç sürücüsüne görsel arayüz aracılığı ile sunulduğu sistem. Projede Teknik Yönetici sorumluluğu üstlendim.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Python 3
- C#
- PowerVision Configuration Studio
- CAN Haberleşme Hattı

#### ❖ CTIS (Merkezi Lastik Şişirme Sistemi)

#### ASİSGUARD / İZMİR

*Açıklama: Zırhlı Araçlar için geliştirilen yol durumuna göre tekerlerin hava basınçlarını ayarlayan ve CAN haberleşme hattından haberleşme sağlayan kinematik sistem. Projenin gömülü yazılım geliştirici rolünde yer aldım.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- C Programlama Dili
- CAN Haberleşme Hattı

#### ❖ 360 Derece Görüş Sistemi (Panoramik Görüntü Oluşturma) ASİSGUARD / İZMİR

*Açıklama: Herhangi bir ortama yerleştirilen kameralardan gelen görüntülerin işlenerek Panoramik Görüntü elde edilmesi projesidir.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Python 3
- OpenCV
- C++
- Cuda

#### ❖ Füzyon Projesi

#### ASİSGUARD / İZMİR

*Açıklama: Termal ve Normal kameradan gelen görüntülerin işlenerek çıkarılması ve sonrasında ısı haritasının ortaya çıkartılması. Ortaya çıkan ısı haritası kullanıcının programa bir arayüz aracılığı ile sunduğu parametreler aracılığıyla dinamik olarak değişmektedir. Proje için Windows ve Linux işletim sisteminde çalışabilecek bir uygulama geliştirilmiştir.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Python 3
- QT

- OpenCV

- Cuda

#### ❖ Otonom Araç Projesi

#### ASISGUARD / İZMİR

*Açıklama: Otonom sürüş kapsamında yürütülen Arge projesidir. Proje kapsamında Donkey Car frameworku üzerinde çalışma yapılmıştır. Derin öğrenme yöntemlerinin kullanıldığı altyapı görüntü işleme teknikleri ile desteklenmiş ve otonom sürüş işleminin başarısı simülasyon üzerinden takip edilmiştir.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- Python 3
- OpenCV
- Donkey Car
- Keras

#### ❖ BAP Bilgi Sistemi

#### UNİPA / İZMİR

*Açıklama: Öğretim Üyelerinin Üniversite içi/dışı proje bilgi girişlerinin gerçekleştirildiği sistemdir. Proje içerisinde proje yürütücülüğünün yanında yazılım geliştirme, analiz ve test süreçlerinde yer aldım.*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- .NET CORE
- MS-SQL
- GO
- Ocelot
- Javascript
- JQuery
- Python3
- Hangfire

#### ❖ Akademik Veri Yönetim Sistemi

#### UNİPA / İZMİR

*Açıklama: Öğretim Üyelerinin Üniversite içi/dışı proje bilgi girişlerinin gerçekleştirildiği sistemdir. Proje içerisinde proje yürütücülüğünün yanında yazılım geliştirme, analiz ve test süreçlerinde yer aldım .*

*Kullanılan Teknolojiler ve Programlama Dilleri:*

- .NET CORE
- MS-SQL
- GO
- Ocelot
- Javascript
- JQuery
- Python3
- Hangfire



**EKLER**

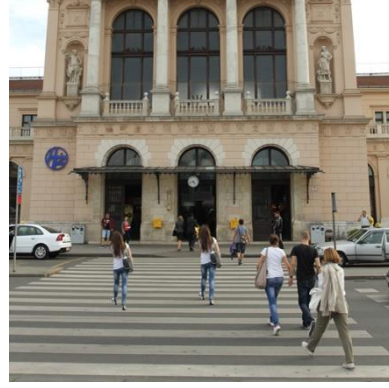
Ek 1 İdeal Parametreler Kullanılarak Üzerinde Çalışılan Görüntüler  
ve İşlenmiş Halleri



## Ek 1 İdeal Parametreler Kullanılarak Üzerinde Çalışılan Görüntüler ve İşlenmiş Halleri



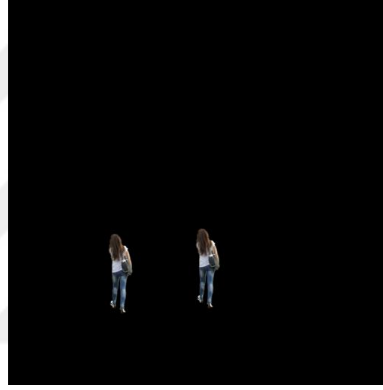
(a)



(b)



(c)



(d)

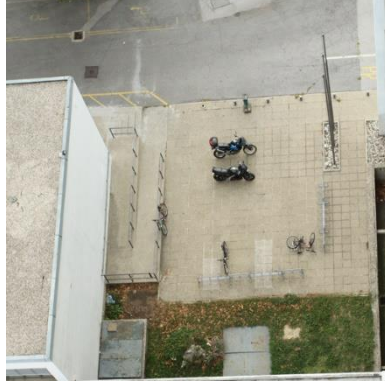


(e)

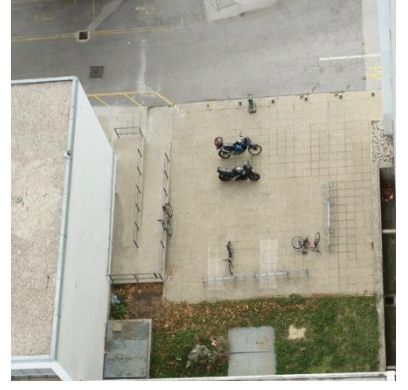


(f)

Şekil A.1 COMOFOFOD-1 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları (d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



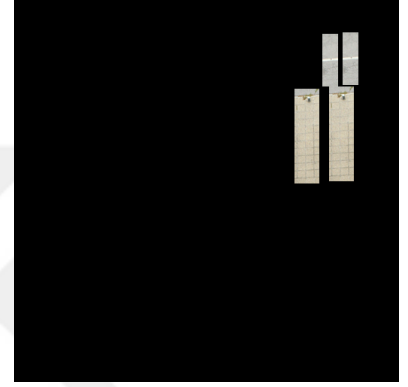
(a)



(b)



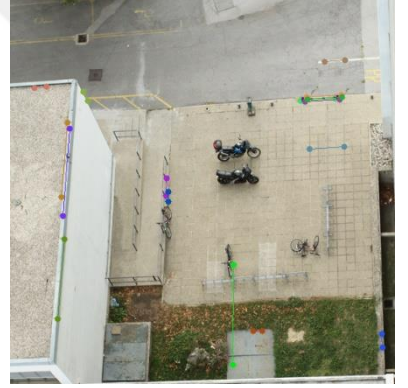
(c)



(d)



(e)



(f)

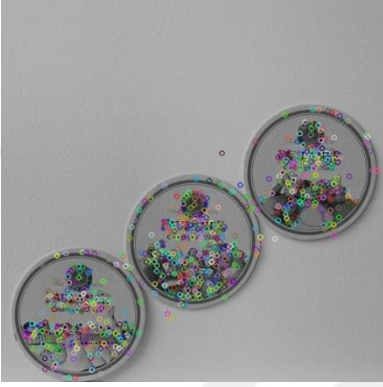
Şekil A.2 COMOFOD-2 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları (d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



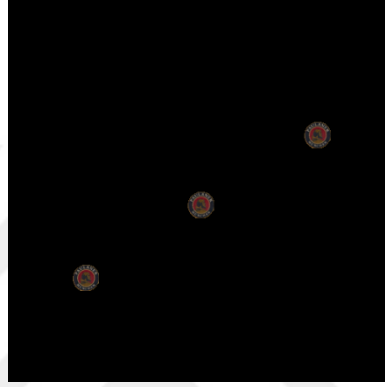
(a)



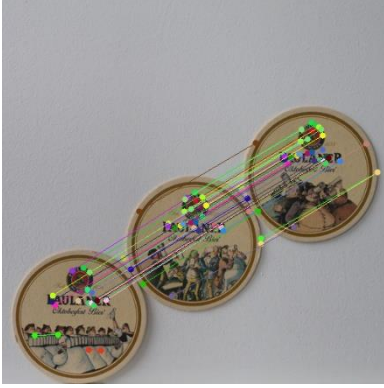
(b)



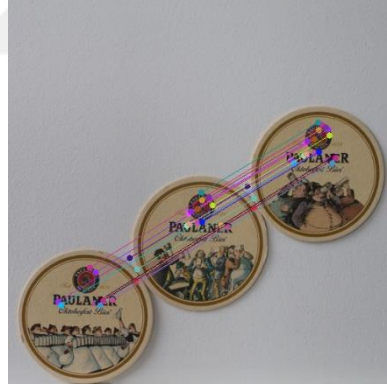
(c)



(d)



(e)



(f)

Şekil A.3 COMOFOOD-4 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları (d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



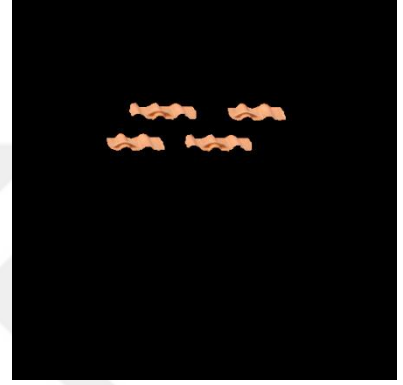
(a)



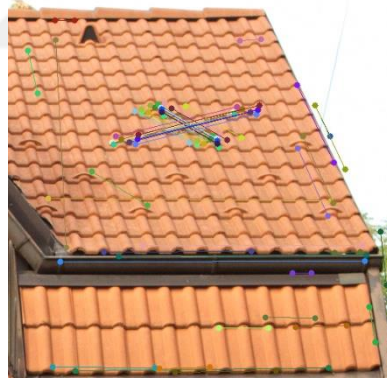
(b)



(c)



(d)



(e)



(f)

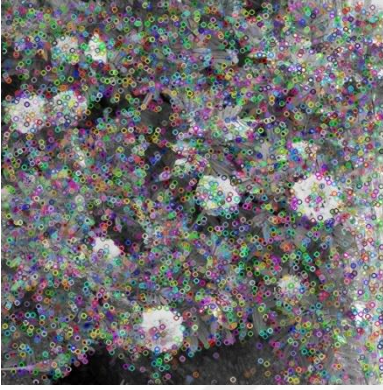
Şekil A.4 COMOFOD-7 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları (d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



(b)



(c)



(d)



(e)



(f)

Şekil A.5 COMOFOOD-11 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



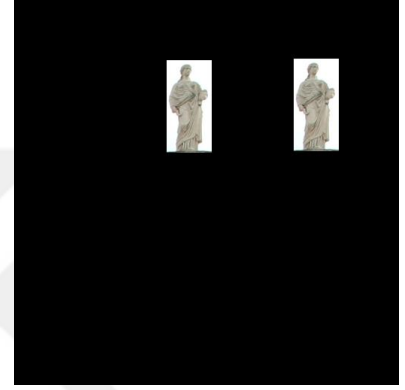
(a)



(b)



(c)



(d)



(e)

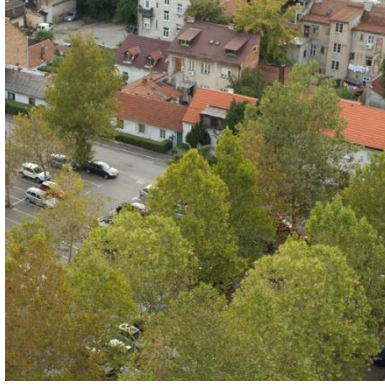


(f)

Şekil A.6 COMOFOFOD-12 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



(b)



(c)



(d)

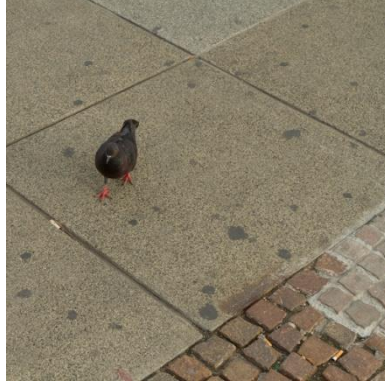


(e)

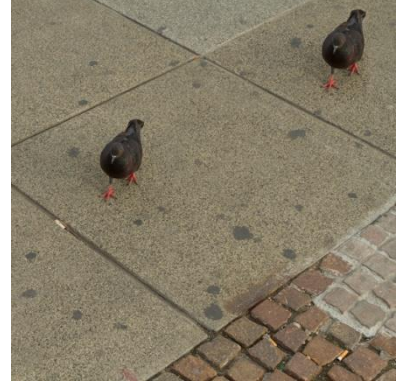


(f)

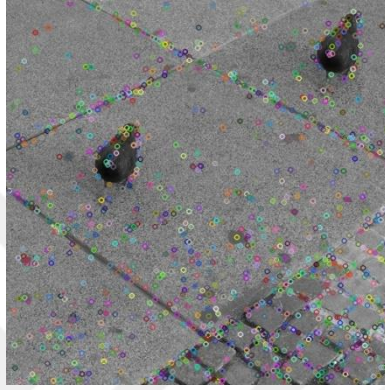
Şekil A.7 COMOFOOD-13 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



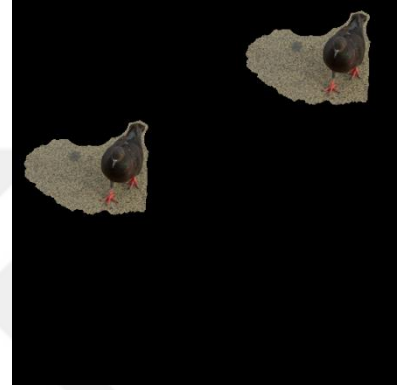
(a)



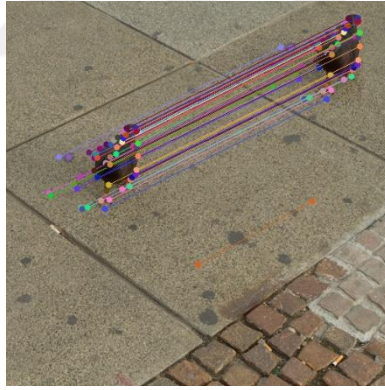
(b)



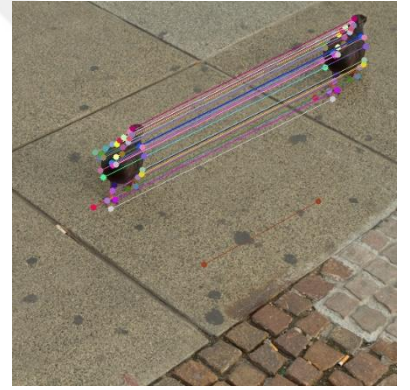
(c)



(d)



(e)



(f)

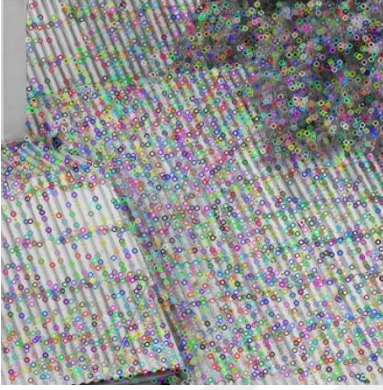
Şekil A.8 COMOFOOD-15 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



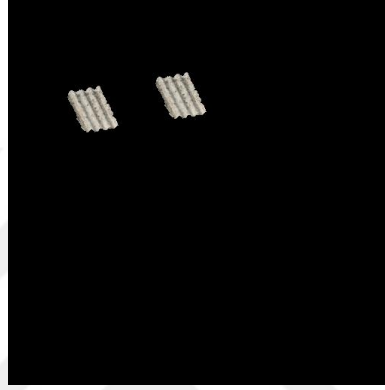
(a)



(b)



(c)



(d)



(e)

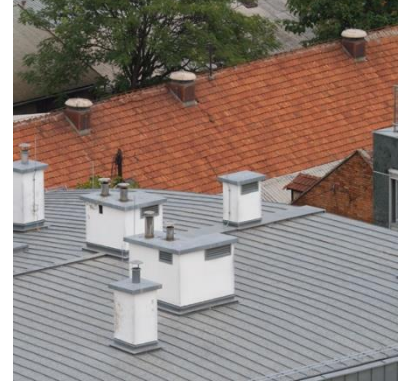


(f)

Şekil A.9 COMOFOOD-16 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



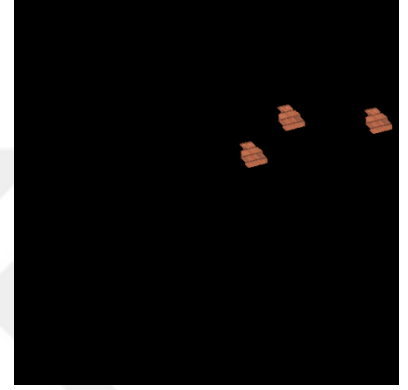
(a)



(b)



(c)



(d)



(e)



(f)

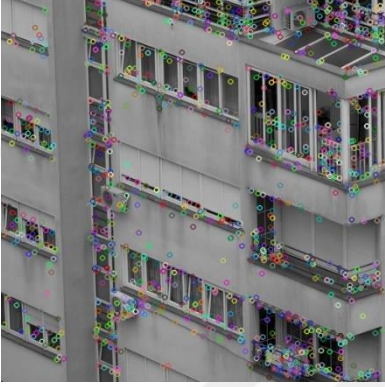
Şekil A.10 COMOFOD-22 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



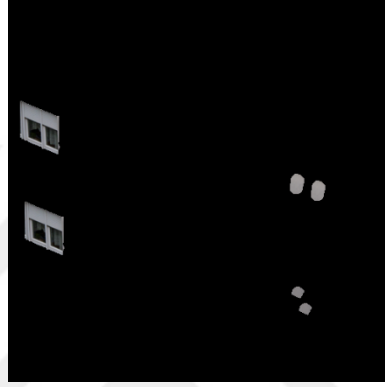
(a)



(b)



(c)



(d)



(e)



(f)

Şekil A.11 COMOFOD-23 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



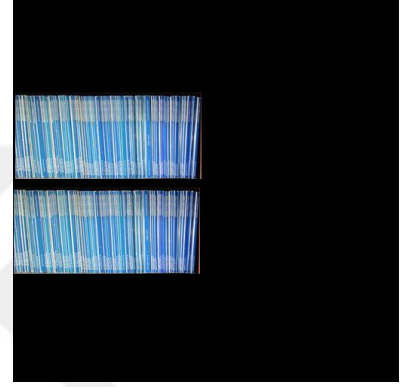
(a)



(b)



(c)



(d)

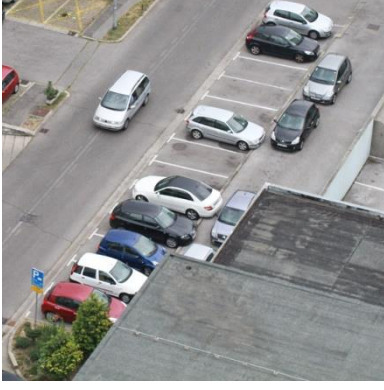


(e)

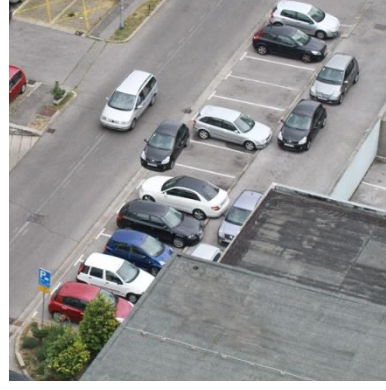


(f)

Şekil A.12 COMOFOD-24 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



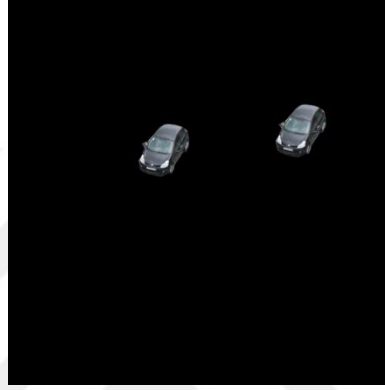
(a)



(b)



(c)



(d)



(e)



(f)

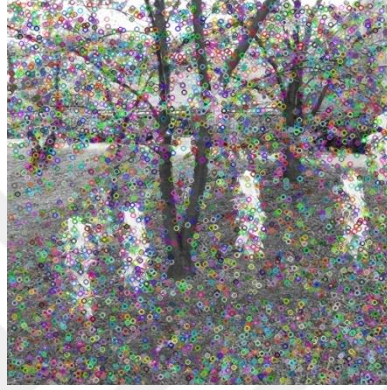
Şekil A.13 COMOFOD-26 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



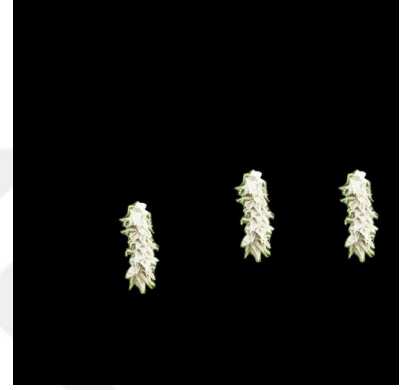
(a)



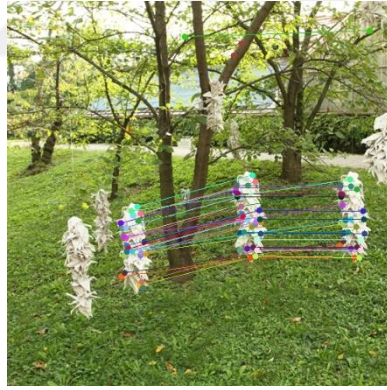
(b)



(c)



(d)

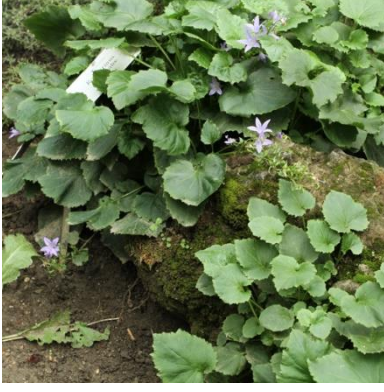


(e)



(f)

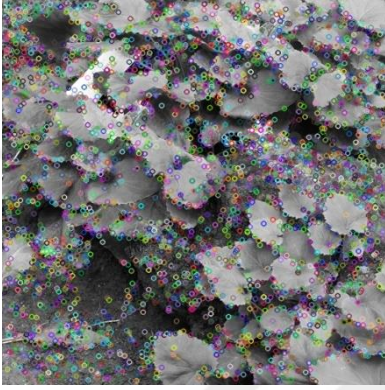
Şekil A.14 COMOFOD-27 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



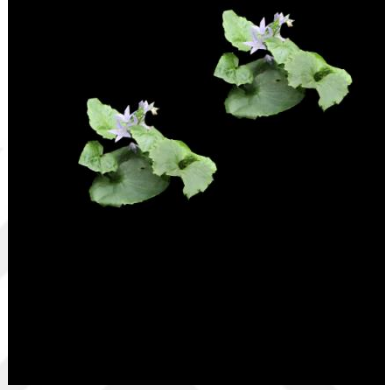
(a)



(b)



(c)



(d)



(e)



(f)

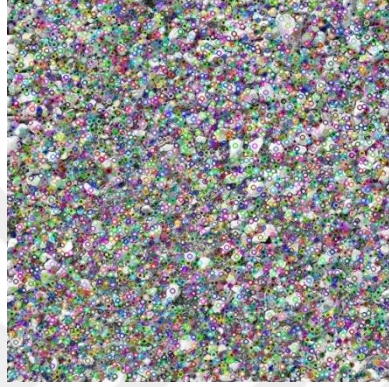
Şekil A.15 COMOFOD-28 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



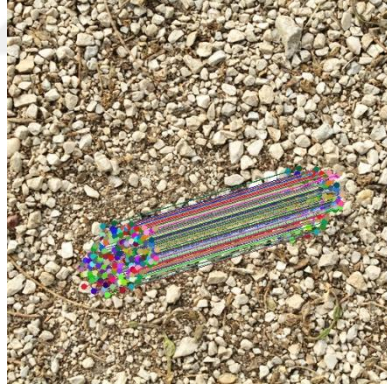
(b)



(c)



(d)

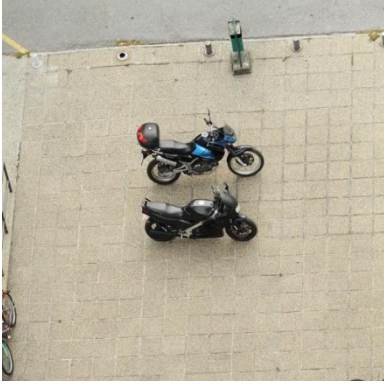


(e)

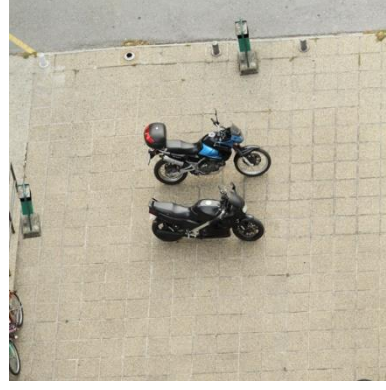


(f)

Şekil A.16 COMOFOD-29 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



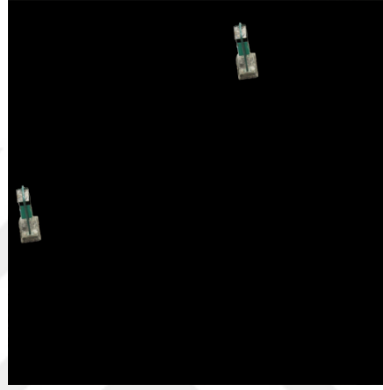
(a)



(b)



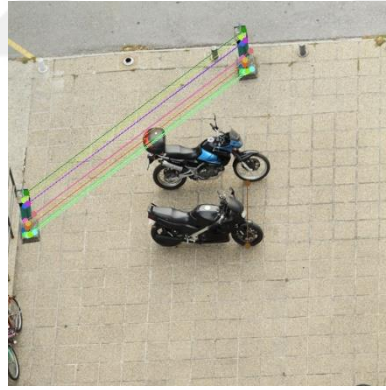
(c)



(d)



(e)

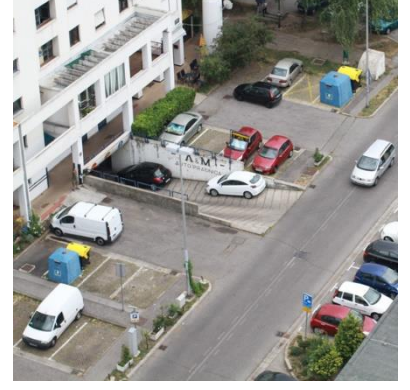


(f)

Şekil A.17 COMOFOD-31 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



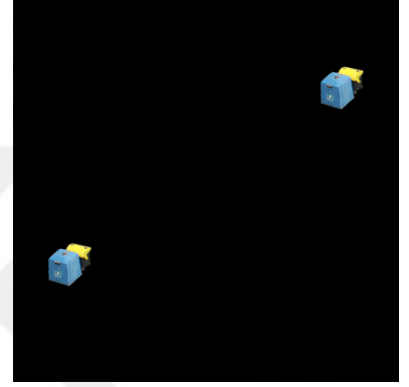
(a)



(b)



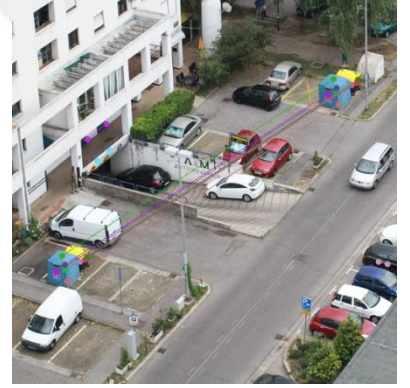
(c)



(d)



(e)



(f)

Şekil A.18 COMOFOOD-72 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



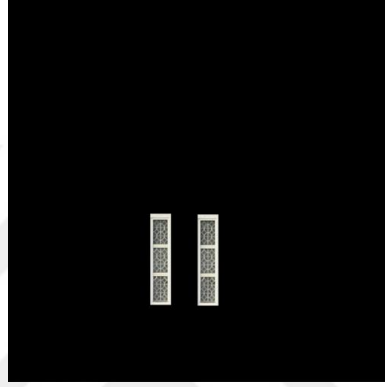
(a)



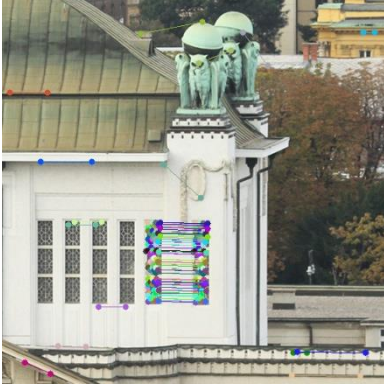
(b)



(c)



(d)

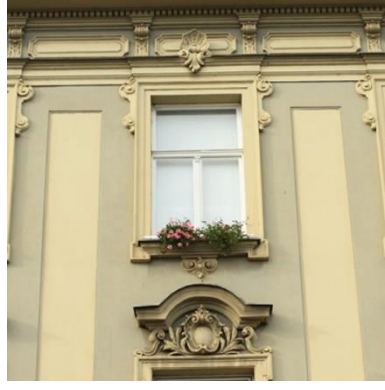


(e)

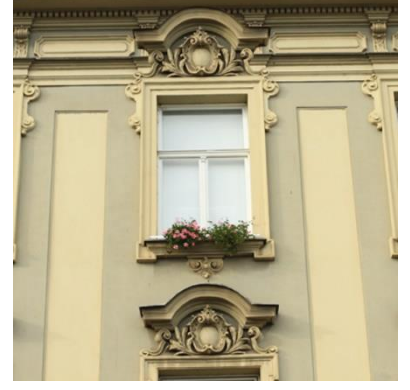


(f)

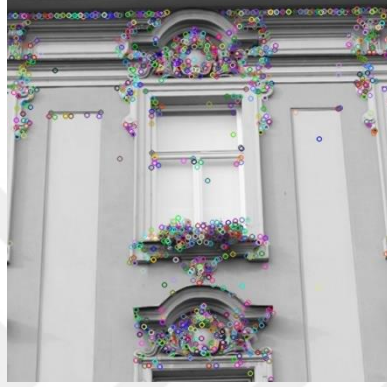
Şekil A.19 COMOFOD-79 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



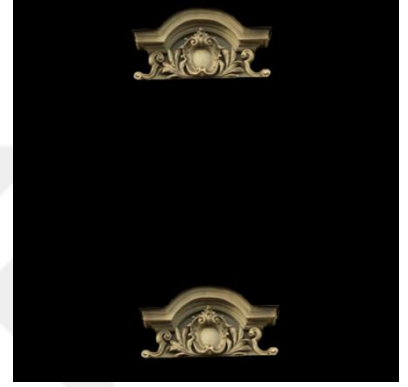
(a)



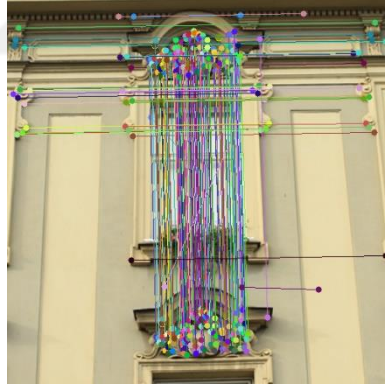
(b)



(c)



(d)



(e)



(f)

Şekil A.20 COMOFOFOD-114 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



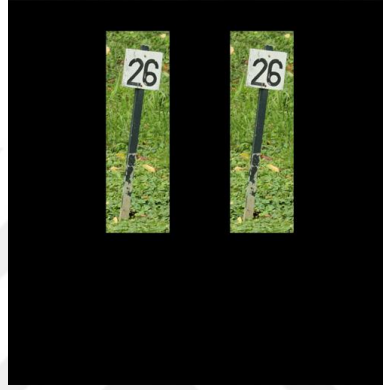
(a)



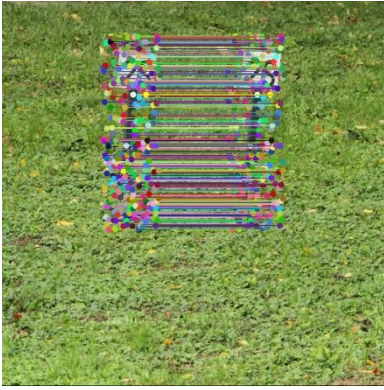
(b)



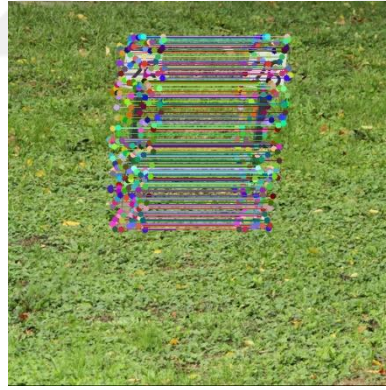
(c)



(d)

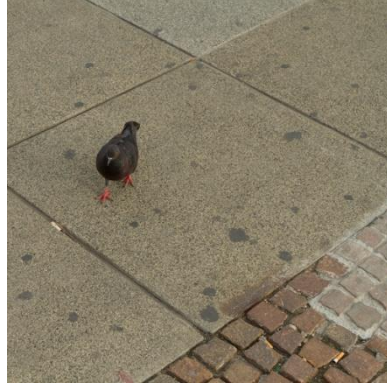


(e)

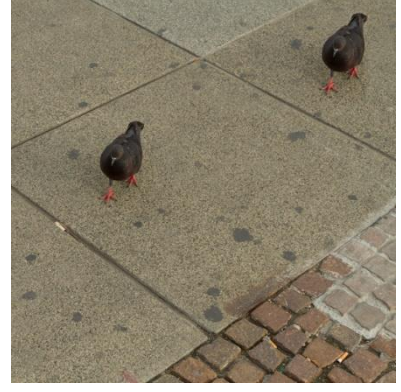


(f)

Şekil A.21 COMOFOD-75 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



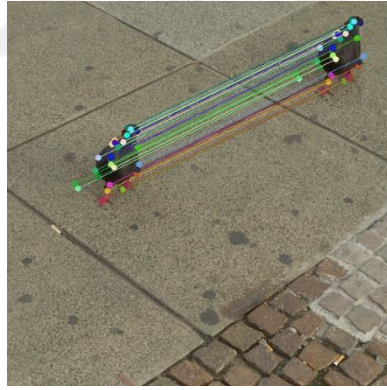
(b)



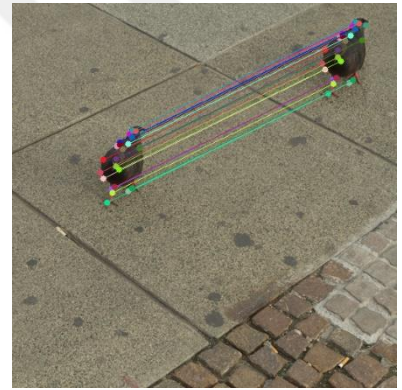
(c)



(d)

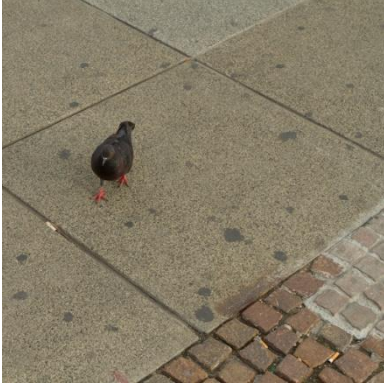


(e)

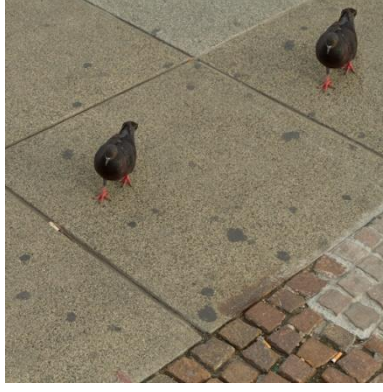


(f)

Şekil A.22 COMOFOOD-50 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



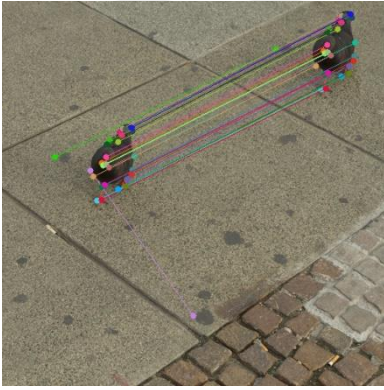
(b)



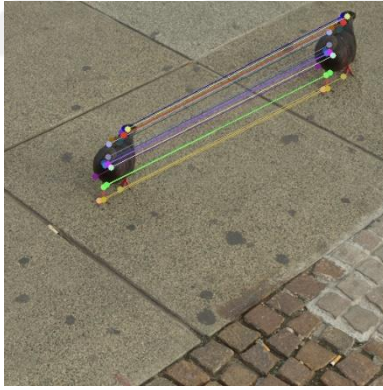
(c)



(d)



(e)



(f)

Şekil A.23 COMOFOD-51 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



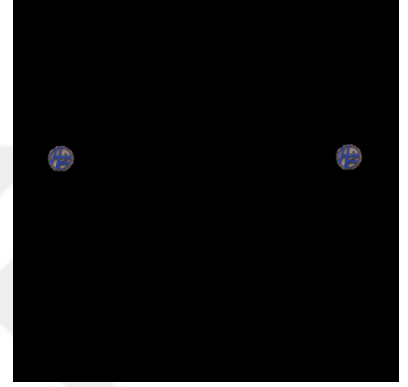
(a)



(b)



(c)



(d)

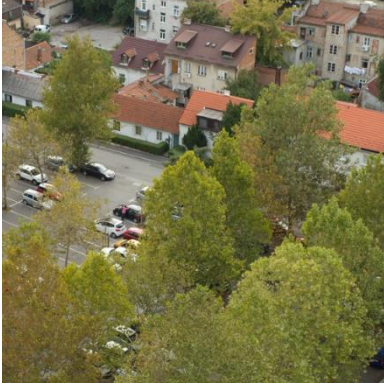


(e)

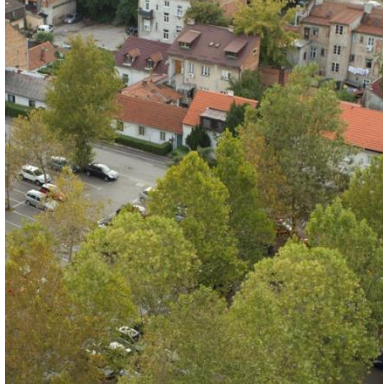


(f)

Şekil A.24 COMOFOOD-141 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu



(a)



(b)



(c)



(d)



(e)



(f)

Şekil A.25 COMOFOOD-167 (a) Referans (b) Sahteciliğe Uğramış Resim (c) SIFT noktaları  
(d) Kopyalanan alan (e) Eşleme Sonucu (f) RANSAC Sonucu

## Ek 2 Tez Çalışması Kaynak Kodları

# Tez çalışmasının arayüzünü hazırlayan .py dosyası (ApplicationQT.py) #

```
from PyQt5 import QtCore, QtGui, QtWidgets
import sys
import os
import CornerDetection.CornerDetectionModule as cdm
import Utilities.FileOperations as fo

import FeatureMatchers.Sift_Daisy_KNN as sdknn
import FeatureMatchers.Sift_Sift_KNN as ssknn
import FeatureMatchers.Sift_Surf_KNN as ssuknn
import FeatureMatchers.Sift_Orb_KNN as soknn
import FeatureMatchers.Sift_Freak_KNN as sfknn
import FeatureMatchers.Sift_Fatih_LBP as flbp
from FeatureMatchers import VBSD
import FeatureMatchers.Sift_RotationOrientedLBP as rol
import FeatureMatchers.Sift_LBP_KNN as olbp

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(366, 312)
        self.commandLinkButton = QtWidgets.QCommandLinkButton(Form)
        self.commandLinkButton.setGeometry(QtCore.QRect(10, 260, 185, 41))
        self.commandLinkButton.setObjectName("commandLinkButton")
        self.widget = QtWidgets.QWidget(Form)
        self.widget.setGeometry(QtCore.QRect(10, 10, 351, 25))
        self.widget.setObjectName("widget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.widget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.btnBrowseFile = QtWidgets.QPushButton(self.widget)
        self.btnBrowseFile.setObjectName("btnBrowseFile")
        self.horizontalLayout.addWidget(self.btnBrowseFile)
        self.btnProcess = QtWidgets.QPushButton(self.widget)
        self.btnProcess.setObjectName("btnProcess")
        self.horizontalLayout.addWidget(self.btnProcess)
        self.widget1 = QtWidgets.QWidget(Form)
        self.widget1.setGeometry(QtCore.QRect(10, 50, 351, 163))
        self.widget1.setObjectName("widget1")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.widget1)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.label = QtWidgets.QLabel(self.widget1)
        self.label.setObjectName("label")
        self.horizontalLayout_2.addWidget(self.label)
        self.cmbCornerDetection = QtWidgets.QComboBox(self.widget1)
        self.cmbCornerDetection.setObjectName("cmbCornerDetection")
        self.horizontalLayout_2.addWidget(self.cmbCornerDetection)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.label_2 = QtWidgets.QLabel(self.widget1)
        self.label_2.setObjectName("label_2")
        self.horizontalLayout_3.addWidget(self.label_2)
        self.cmbFeatureMatcher = QtWidgets.QComboBox(self.widget1)
        self.cmbFeatureMatcher.setObjectName("cmbFeatureMatcher")
        self.horizontalLayout_3.addWidget(self.cmbFeatureMatcher)
```

```

self.verticalLayout.addLayout(self.horizontalLayout_3)
self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
self.horizontalLayout_4.setObjectName("horizontalLayout_4")
self.label_3 = QtWidgets.QLabel(self.widget1)
self.label_3.setObjectName("label_3")
self.horizontalLayout_4.addWidget(self.label_3)
self.dspNNThreshold = QtWidgets.QDoubleSpinBox(self.widget1)
self.dspNNThreshold.setObjectName("dspNNThreshold")
self.horizontalLayout_4.addWidget(self.dspNNThreshold)
self.verticalLayout.addLayout(self.horizontalLayout_4)
self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
self.horizontalLayout_5.setObjectName("horizontalLayout_5")
self.label_4 = QtWidgets.QLabel(self.widget1)
self.label_4.setObjectName("label_4")
self.horizontalLayout_5.addWidget(self.label_4)
self.spMinimumDistance = QtWidgets.QSpinBox(self.widget1)
self.spMinimumDistance.setObjectName("spMinimumDistance")
self.horizontalLayout_5.addWidget(self.spMinimumDistance)
self.verticalLayout.addLayout(self.horizontalLayout_5)
self.horizontalLayout_6 = QtWidgets.QHBoxLayout()
self.horizontalLayout_6.setObjectName("horizontalLayout_6")
self.label_5 = QtWidgets.QLabel(self.widget1)
self.label_5.setObjectName("label_5")
self.horizontalLayout_6.addWidget(self.label_5)
self.spPatchSize = QtWidgets.QSpinBox(self.widget1)
self.spPatchSize.setObjectName("spPatchSize")
self.horizontalLayout_6.addWidget(self.spPatchSize)
self.verticalLayout.addLayout(self.horizontalLayout_6)
self.label.raise_()
self.label.raise_()
self.label.raise_()
self.label.raise_()
self.commandLinkButton.raise_()
self.retranslateUi(Form)
# self.btnProcess.clicked.connect(self.btnProcess.click)
# self.btnBrowseFile.clicked.connect(self.btnBrowseFile.click)
# self.commandLinkButton.clicked.connect(self.commandLinkButton.click)
QtCore.QMetaObject.connectSlotsByName(Form)

```

#### **def retranslateUi(self, Form):**

```

_translate = QtCore.QCoreApplication.translate
Form.setWindowTitle(_translate("Form", "Copy Move Forgery Detector"))
self.commandLinkButton.setText(_translate("Form", "Open CMFD folder"))
self.btnBrowseFile.setText(_translate("Form", "Browse File"))
self.btnProcess.setText(_translate("Form", "Process"))
self.label.setText(_translate("Form", "<html><head/><body><p><span
weight:600;\>Corner Detection Algorithms</span></p></body></html>"))
self.label_2.setText(_translate("Form", "<html><head/><body><p><span
weight:600;\>Feature Matcher</span></p></body></html>"))
self.label_3.setText(_translate("Form", "<html><head/><body><p><span
weight:600;\>NN threshold</span></p></body></html>"))
self.label_4.setText(_translate("Form", "<html><head/><body><p><span
weight:600;\>Minimum Distance</span></p></body></html>"))
self.label_5.setText(_translate("Form", "<html><head/><body><p><span
weight:600;\>Patch Size</span></p></body></html>"))

```

```
class Prog(QtWidgets.QMainWindow):
```

```
    def __init__(self):
```

```
        self.listeForCombobox = ["Shi-Tomasi", "Harris", "SIFT"]
        self.listeForCombobox_fm = { "sift_daisy_knn" : "SIFT_DAISSY_KNN",
                                     "sift_daisy_knn" : "SIFT_DAISSY_KNN",
                                     "sift_sift_knn" : "SIFT_SIFT_KNN",
                                     "sift_surf_knn" : "SIFT_SURF_KNN",
                                     "sift_orb_knn" : "SIFT_ORB_KNN",
                                     "sift_freak_knn" : "SIFT_FREAK_KNN",
                                     "vbsd" : "VBSD",
                                     "sift_rol" : "SIFT_Rotation_Dominant_LBP",
                                     "sift_new_lbp" : "SIFT_MSRO_LBP",
                                     "sift_org_lbp" : "SIFT_ORG_LBP",
                                     "sift_clbp" : "SIFT_CLBP",
                                     }
        self.result_string = "{0}\CMFD_Results".format(os.environ['USERPROFILE'])
        super().__init__()
        self.ui = Ui_Form()
        self.ui.setupUi(self)
        #item assignments
        self.ui.cmbCornerDetection.addItem(self.listeForCombobox)
        self.ui.cmbFeatureMatcher.addItem(list(self.listeForCombobox_fm.values()))
        self.selectedFeatureMatcher = self.ui.cmbFeatureMatcher.currentText()
        self.ui.dspNNThreshold.setValue(0.75)
        self.ui.spPatchSize.setValue(15)
        self.ui.spMinimumDistance.setValue(12)
        #events
        self.ui.btnBrowseFile.clicked.connect(self.openFile)
        self.ui.btnProcess.clicked.connect(self.process)
        self.ui.cmbFeatureMatcher.currentIndexChanged.connect(self.setFeatureMatcher)
        self.ui.commandLinkButton.clicked.connect(self.openCMFDShortcut)
```

```
    def openCMFDShortcut(self):
```

```
        # subprocess.Popen(r'explorer /select,{0}_Images'.format(url))
        if not os.path.exists(self.result_string):
            os.mkdir(self.result_string)
        openFilePath = '{0}/{1}_Images'.format(self.result_string, self.selectedFeatureMatcher)
        if not os.path.exists(openFilePath):
            os.mkdir(openFilePath)
        os.startfile(openFilePath)
```

```
    def openFile(self):
```

```
        file_dialog = QtWidgets.QFileDialog()
        file_dialog.setFileMode(QtWidgets.QFileDialog.AnyFile)
        file_dialog.setNameFilter("Images(*.jpg *.gif *.bmp *.png *.tiff *.jpeg);;CoMoFod (*.F.*)")
        file_dialog.setOption(QtWidgets.QFileDialog.DontUseNativeDialog, True)
        file_view = file_dialog.findChild(QtWidgets.QListView, 'listView')
```

```
        if file_view:
```

```
            file_view.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)
```

```
        f_tree_view = file_dialog.findChild(QtWidgets.QTreeView)
```

```
        if f_tree_view:
```

```
            f_tree_view.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)
```

```
        if file_dialog.exec():
```

```
            paths = file_dialog.selectedFiles()
```

```
            self.path_for_image = paths
```

```
    def setFeatureMatcher(self):
```

```
        self.selectedFeatureMatcher = self.ui.cmbFeatureMatcher.currentText()
```

```
def popup_warning(self, msg):
    title = "WARNING"
    QtWidgets.QMessageBox.warning(self, title, msg, QtWidgets.QMessageBox.Close)
```

```
def popup_error(self, msg):
    title = "ERROR"
    QtWidgets.QMessageBox.critical(self, title, msg, QtWidgets.QMessageBox.Close)
```

```
def popup_information(self, msg):
    title = "Info"
    QtWidgets.QMessageBox.information(self, title, msg)
```

```
def process(self):
    try:
```

```
        if self.path_for_image is None:
            self.popup_warning("Copy Move Forgery Detection.", "Dosya Seçimi Yapınız.")
            return
        if not os.path.exists(self.result_string):
            os.mkdir(self.result_string)
        nearestNeighbourDistanceThreshold = None
        minimumDistance = None
        nearestNeighbourDistanceThreshold = self.ui.dspNNThreshold.value()
        minimumDistance = self.ui.spMinimumDistance.value()
        patchSize = self.ui.spPatchSize.value()
        featureListim = []
        if self.selectedFeatureMatcher == self.listeForCombobox_fm["vbsd"]:
            path = "{0}\{1}_Images".format(self.result_string, self.listeForCombobox_fm["vbsd"])
            if not os.path.exists(path):
                os.mkdir(path)
            for selectedResim in self.path_for_image:
                featureListim, mag_ori_image = cdm.Sift(selectedResim, patchSize, path)
                VBSD.VBSD_FeaturelariOlustur(featureListim, 260, 5)
                VBSD.VBSD_DictionaryEslestir(featureListim, 4, 10)
                VBSD.VBSD_cmfdSonuclariniDosyayaYaz(selectedResim, path,
featureListim, nearestNeighbourDistanceThreshold)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_daisy_knn"] :
                sdknn.sift_daisy_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_daisy_knn"]),
                    nearestNeighbourDistanceThreshold, minimumDistance)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_sift_knn"] :
                ssknn.sift_sift_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_sift_knn"]),
                    nearestNeighbourDistanceThreshold, minimumDistance)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_surf_knn"] :
                ssuknn.sift_surf_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_surf_knn"]),
                    nearestNeighbourDistanceThreshold, minimumDistance)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_orb_knn"] :
                soknn.sift_orb_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_orb_knn"]),
                    nearestNeighbourDistanceThreshold,
                    minimumDistance)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_freak_knn"] :
                sfknn.sift_freak_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_freak_knn"]),
                    nearestNeighbourDistanceThreshold,
                    minimumDistance)
            elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_rol"] :
                rol.sift_lbp(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
self.listeForCombobox_fm["sift_rol"]),
                    nearestNeighbourDistanceThreshold,
                    minimumDistance, ThresholdForGLCM=None)
```

```

elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_new_lbp"]:
    flbp.sift_lbp(self.path_for_image, patchSize,
        "{0}\{1}_Images".format(self.result_string, self.listeForCombobox_fm["sift_new_lbp"]),
        nearestNeighbourDistanceThreshold,
        minimumDistance, ThresholdForGLCM=None)
elif self.selectedFeatureMatcher == self.listeForCombobox_fm["sift_org_lbp"]:
    olbp.sift_olbp_knn(self.path_for_image, patchSize, "{0}\{1}_Images".format(self.result_string,
        self.listeForCombobox_fm[
            "sift_org_lbp"]),
        nearestNeighbourDistanceThreshold, minimumDistance)

# endregion

self.popup_information("İşlem Sonlandı")
except BaseException as e:
    self.popup_error("Hata Olustu {0}".format(str(e)))
pass

if __name__ == "__main__":
    Program = QtWidgets.QApplication(sys.argv)
    MyProg = Prog()
    MyProg.show()
    sys.exit(Program.exec_())

# Kopyala taşı sahtecilik istatistiklerini hazırlayan .py dosyası (Validation.py) #

import cv2
import random

''' Validasyon modulu
    Toplam Eslesme Sayisi -> Her noktayı eslestirdigimiz icin bir sikinti yok.
    Toplam Eslesme Sayisi = Eslesmeye Dahil Edilmemis ( Zero + White ) + Ransacsiz ( Zero + White )
    Toplam Maskedeki beyaz noktaların sayısı = Ransacsiz Esleme Sayisi + Eslesmeye Dahil Edilmemis Maske
    White Esleme Sayisi
    Ransacli Maske white -> RANSAC resminde Maskedeki beyaz degerlerin sayisini veriyor
    Ransacsiz Maske white -> RANSACsiz resimdeki Maskedeki beyaz degerlerin sayisini veriyor
    Ransacsiz Maske Zero -> RANSACsiz resimdeki Maskedeki siyah degerlerin sayisini veriyor
    Ransacli Maske Zero -> RANSAC resminde Maskedeki siyah degerlerin sayisini veriyor
'''

def create_validation_file(directory_name, total_sec):
    file_name = "{0}\\Validation_{1}.txt".format(directory_name, total_sec)
    f = open(file_name, 'w')
    f.close()
    return file_name

def validate(validation_directory_name, validation_normal, validation_ransac,
    validation_image_binary_path, resim_index_arr, masked_image_path, validation_silinmis,
    validation_total_match,
    directory_name):
    try:
        validation_image = cv2.imread(validation_image_binary_path)
        masked_image_ransac = cv2.imread(masked_image_path) # masked image uzerinde cizme yapcam.
        masked_image_ransacsiz = masked_image_ransac.copy()
        ayrac = ":"
        maske_white_nokta_sayisi = 0
        maske_zero_nokta_sayisi = 0
        ransacsiz_esleme_sayisi = len(validation_normal)

```

```

ransaccli_esleme_sayisi = len(validation_ransac)
ransacsiz_maske_white_esleme_sayisi = 0
ransaccli_maske_white_esleme_sayisi = 0
ransacsiz_maske_zero_esleme_sayisi = 0
ransaccli_maske_zero_esleme_sayisi = 0
ransaccli_maske_white_dogru_esleme_sayisi = 0 # bunu teker teker kontrol edicem.
ransacsiz_maske_white_dogru_esleme_sayisi = 0 # bunu teker teker kontrol edicem.

silinmis_maske_white = 0
silinmis_maske_zero = 0

validation_binary_image_zeros = validation_image[:, :, 0]
hei, wi = validation_binary_image_zeros.shape
for h in range(hei):
    for w in range(wi):
        color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
        deger = validation_binary_image_zeros[h, w]
        if deger == 0: #maske zero degerler artacak
            maske_zero_nokta_sayisi += 1
            check_normal = [x for x in validation_normal if x[0] == h and x[1] == w]
            if len(check_normal) > 0:
                ransacsiz_maske_zero_esleme_sayisi += len(check_normal)
                check_ransac = [x for x in validation_ransac if x[0] == h and x[1] == w]
                if len(check_ransac) > 0:
                    ransaccli_maske_zero_esleme_sayisi += len(check_ransac)
                silinmis_check_normal = [x for x in validation_silinmis if x[0] == h and x[1] == w]
                if len(silinmis_check_normal) > 0:
                    silinmis_maske_zero += len(silinmis_check_normal)
            else:
                maske_white_nokta_sayisi += 1
                check_normal = [x for x in validation_normal if x[0] == h and x[1] == w]
                if len(check_normal) > 0:
                    ransacsiz_maske_white_esleme_sayisi += len(check_normal)
                    connect_points(masked_image_ransacsiz, check_normal, color)
                check_ransac = [x for x in validation_ransac if x[0] == h and x[1] == w]
                if len(check_ransac) > 0:
                    ransaccli_maske_white_esleme_sayisi += len(check_ransac)
                    connect_points(masked_image_ransaccli, check_ransac, color)
                #degeri 255 olup silinmis olabilir, ondan onlarida tutmam gerekiyor
                silinmis_check_normal = [x for x in validation_silinmis if x[0] == h and x[1] == w]
                if len(silinmis_check_normal) > 0:
                    silinmis_maske_white += len(silinmis_check_normal)
str_dosya_indeksi = ". ".join(resim_index_arr)
cv2.imwrite('{0}\Masked_image_ransaccli_{1}'.format(directory_name, str_dosya_indeksi),
masked_image_ransaccli)
cv2.imwrite('{0}\Masked_image_ransacsiz_{1}'.format(directory_name, str_dosya_indeksi),
masked_image_ransacsiz)
str_dosya_ransacsiz_esleme_sayisi = "Ransacsiz Esleme Sayisi{0}{1}\n".format(ayrac,
ransacsiz_esleme_sayisi)
str_dosya_ransaccli_esleme_sayisi = "Ransaccli Esleme Sayisi{0}{1}\n".format(ayrac,
ransaccli_esleme_sayisi)
str_ransaccli_maske_white_esleme_sayisi = "Ransaccli Maske white Esleme Sayisi{0}{1}\n".format(ayrac,
ransaccli_maske_white_esleme_sayisi)
str_ransacsiz_maske_white_esleme_sayisi = "Ransacsiz Maske white Esleme
Sayisi{0}{1}\n".format(ayrac, ransacsiz_maske_white_esleme_sayisi)
str_ransacsiz_maske_zero_esleme_sayisi = "Ransacsiz Maske Zero Esleme
Sayisi{0}{1}\n".format(ayrac, ransacsiz_maske_zero_esleme_sayisi)
str_ransaccli_maske_zero_esleme_sayisi = "Ransaccli Maske Zero Esleme Sayisi{0}{1}\n".format(ayrac,
ransaccli_maske_zero_esleme_sayisi)
str_ransaccli_maske_white_dogru_esleme_sayisi = "Ransaccli Maske White Dogru Esleme
Sayisi{0}{1}\n".format(ayrac, ransaccli_maske_white_dogru_esleme_sayisi)
str_ransacsiz_maske_white_dogru_esleme_sayisi = "Ransacsiz Maske White Dogru Esleme
Sayisi{0}{1}\n".format(ayrac,
ransacsiz_maske_white_dogru_esleme_sayisi)

```

```

        str_silinmis_maske_white = "Eslesmeye Dahil Edilmemis Maske White Esleme
Sayisi{0}{1}\n".format(ayrac, silinmis_maske_white)
        str_silinmis_maske_zero = "Eslesmeye Dahil Edilmemis Maske Zero Esleme
Sayisi{0}{1}\n".format(ayrac, silinmis_maske_zero)
        str_validation_total_match_zero = "Toplam Esleme Sayisi{0}{1}\n".format(ayrac,
validation_total_match)

```

```

        f = open(validation_directory_name, "a+")
        try:
            f.write("Dosya Indeksi{0}{1}\r\n".format(ayrac, str_dosya_indeksi))
            f.write(str_dosya_ransacsiz_esleme_sayisi)
            f.write(str_dosya_ransaccli_esleme_sayisi)
            f.write(str_ransaccli_maske_white_esleme_sayisi)
            f.write(str_ransacsiz_maske_white_esleme_sayisi)
            f.write(str_ransaccli_maske_zero_esleme_sayisi)
            f.write(str_ransacsiz_maske_zero_esleme_sayisi)
            f.write(str_ransaccli_maske_white_dogr_u_esleme_sayisi)
            f.write(str_ransacsiz_maske_white_dogr_u_esleme_sayisi)
            f.write(str_silinmis_maske_white)
            f.write(str_silinmis_maske_zero)
            f.write(str_validation_total_match_zero)

        f.write("*****\n")
    )
    f.close()
except BaseException as e:
    f.close()
    print("Validation Dosya Yazma Hata {0} - {1}".format(str(e), "_".join(resim_index_arr)))
    pass

except BaseException as e:
    print("Validation Hata {0} - {1}".format(str(e), "_".join(resim_index_arr)))
    pass

```

**def connect\_points(image, values, color):**

```

    for v in values:
        hei_1, wi_1, hei_2, wi_2 = v
        cv2.circle(image, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
        cv2.circle(image, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
        cv2.line(image, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))

```

**# Feature sınıfı (cs\_Feature.py) #**

**class Feature(object):**

```

    def __init__(self, pointx, pointy, patchValues=None, magnitudeValues=None,
orientationValues=None):
        self.pointX = pointx
        self.pointY = pointy
        self.patchData = patchValues
        self.patchDataMagnitude = magnitudeValues
        self.patchDataOrientation = orientationValues
        self.featureDictionary = {}
        self.featureDictionary_binsize = 0
        self.otherFeatures = []
        self.kontrolText = ""
        self.vektorData = []
        self.glcmTexturelarim = []

```

**def setOtherFeature(self, otherFeaturem):**

```

    self.otherFeature = otherFeaturem

```

```
class Feature_PointsAndValues():
```

```
    """description of class"""
```

```
    def __init__(self, euclidValue, *tupum):
```

```
        self.pointX = tupum[0][0]
```

```
        self.pointY = tupum[0][1]
```

```
        self.featureDictionary_EuclidValue = euclidValue # DictionaryListi yapicam bunu guzel olucak.
```

```
# Köşe noktası tespit modülü (CornerDetectionModule.py) #
```

```
def Sift(path, patchSize, dosyaPath, getMagnituteAndOrientation=False, useGaussBlurFilter=False, calculateLogPolar = False):
```

```
    img = cv2.imread(path)
```

```
    height, width, channel = img.shape
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    sift = cv2.xfeatures2d.SIFT_create()
```

```
    kp = sift.detect(gray, None)
```

```
    cv2.drawKeypoints(gray, kp, img)
```

```
    # cv2.imwrite("gray.jpg",img)
```

```
    if useGaussBlurFilter:
```

```
        gray = cv2.GaussianBlur(gray, (5, 5), sigmaX=0)
```

```
    mag_orient_image = None
```

```
    if getMagnituteAndOrientation:
```

```
        mag_image, orient_image, _ = get_mag_ang(gray)
```

```
    featureList = []
```

```
    yeniPatchSize = int(patchSize / 2)
```

```
    for i in kp:
```

```
        wi = int(i.pt[0])
```

```
        hei = int(i.pt[1])
```

```
    roiMagni = None
```

```
    roiOri = None
```

```
    kontrol = list(filter(lambda x: x.pointX == wi and x.pointY == hei, featureList))
```

```
    if len(kontrol) == 0:
```

```
        if not (yeniPatchSize < wi < width - yeniPatchSize and yeniPatchSize < hei < height - yeniPatchSize):
            continue
```

```
        roi = gray[hei - yeniPatchSize:hei + yeniPatchSize + 1,
                    wi - yeniPatchSize:wi + yeniPatchSize + 1] # roi grayden gelicek
```

```
        if getMagnituteAndOrientation:
```

```
            roiMagni = mag_image[hei - yeniPatchSize:hei + yeniPatchSize + 1,
```

```
                                wi - yeniPatchSize:wi + yeniPatchSize + 1] # roi magnitute gelicek
```

```
            roiOri = orient_image[hei - yeniPatchSize:hei + yeniPatchSize + 1,
```

```
                                wi - yeniPatchSize:wi + yeniPatchSize + 1] # roi magnitute gelicek
```

```
        if calculateLogPolar:
```

```
            roi = cv2.logPolar(roi, (roi.shape[0] / 2, roi.shape[1] / 2),
```

```
                               15,
```

```
                               cv2.WARP_FILL_OUTLIERS) # merkez orta nokta olacak sekilde,radiusu
```

```
    patchsize/2 olacak sekilde ayarladim
```

```
    feature = Feature(wi, hei, roi, roiMagni, orientationValues=roiOri)
```

```
    featureList.append(feature)
```

```
    return featureList, mag_orient_image
```

```
def getGradientMagnitute(im):
```

```
    "Get magnitude of gradient for given image"
```

```
    ddepth = cv2.CV_32F
```

```
    dx = cv2.Sobel(im, ddepth, 1, 0)
```

```
    dy = cv2.Sobel(im, ddepth, 0, 1)
```

```
    dxabs = cv2.convertScaleAbs(dx)
```

```

dyabs = cv2.convertScaleAbs(dy)
mag = cv2.addWeighted(dxabs, 0.5, dyabs, 0.5, 0)
return mag

```

```

def get_mag_ang(img):

```

```

    """
    Gets image gradient (magnitude) and orientation (angle)

```

```

    Args:
        img

```

```

    Returns:
        Gradient, orientation
    """

```

```

    im = np.float32(img) / 255.0
    # Calculate gradient
    gx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=1)
    gy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=1)
    mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees=True)
    return mag, ang, gx, gy

```

```

# DAISY (SIFT_Daisy_KNN.py) #

```

```

import cv2
import numpy as np
from sklearn import linear_model
import os
import random
import datetime
import Validation_Module.Validation as vld

```

```

def sift_daisy_knn(image_file_paths, patchSize, directoryName, thresholdForMinNeighbour=None,
thresholdForMinDistance=None):

```

```

    if not os.path.exists(directoryName):
        os.mkdir(directoryName)
    for p in image_file_paths:
        try:

```

```

            path = p
            img = cv2.imread(path)
            image2 = cv2.imread(path)
            height, width, channel = img.shape

```

```

            resimImageBinaryRansacsiz = np.zeros((height, width, 3), np.uint8)
            resimImageBinaryRansac1 = np.zeros((height, width, 3), np.uint8)
            #resimImageBinaryKeyPoints = np.zeros((height, width, 3), np.uint8)

```

```

            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            sift = cv2.xfeatures2d.SIFT_create()
            kp = sift.detect(gray, None)

```

```

            imgdraw = img.copy()
            graydraw = gray.copy()
            cv2.drawKeypoints(graydraw, kp, imgdraw)
            cv2.imwrite("{0}\SIFT_{1}".format(directoryName, path.split("/")[-1]), imgdraw)

```

```

            daisy = cv2.xfeatures2d.DAISY_create()
            keyPoints, descriptors = daisy.compute(gray, kp)
            keyPoints2 = keyPoints.copy()
            descriptors2 = descriptors.copy()
            bf = cv2.BFMatcher()
            # to get k best matches
            matches = bf.knnMatch(descriptors, descriptors2, k=3)

```

```

good = []
silinecek_indexList = []
validation_total_count = len(matches)
for index in range(0, len(matches)):
    eslesenler = matches[index][1]
    if thresholdForMinDistance is not None:
        kp1 = keyPoints[eslesenler.queryIdx]
        kp2 = keyPoints2[eslesenler.trainIdx]
        # noktaları alıyorum burada
        wi_1 = int(kp1.pt[0])
        hei_1 = int(kp1.pt[1])
        wi_2 = int(kp2.pt[0])
        hei_2 = int(kp2.pt[1])
        uzaklik = ((wi_1-wi_2)**2+(hei_1-hei_2)**2)**0.5
        if uzaklik<= thresholdForMinDistance:
            silinecek_indexList.append(index)
            continue
    if thresholdForMinNeighbour is not None:
        kendisi,m, n = matches[index]
        if m.distance >= thresholdForMinNeighbour * n.distance:
            silinecek_indexList.append(index)
            continue
if len(silinecek_indexList)>0:
    silinecek_indexList = list(set(silinecek_indexList))
    for index in range(0, len(matches)):
        if index not in silinecek_indexList:
            good.append(matches[index])
        else:
            silinecek = matches[index][1]
            kp1 = keyPoints[silinecek.queryIdx]
            kp2 = keyPoints2[silinecek.trainIdx]
            # noktaları alıyorum burada
            wi_s1 = int(kp1.pt[0])
            hei_s1 = int(kp1.pt[1])
            wi_s2 = int(kp2.pt[0])
            hei_s2 = int(kp2.pt[1])

if len(good)>0:
    matches = []
    matches = good

if type(img) is not np.array:
    img = np.array(img)
overlay = img.copy()
ransacImage = img.copy()

alpha = 0.7

source_points = []
destination_points = []

for match in matches:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = match[1]
    kp1 = keyPoints[eslesenler.queryIdx]
    kp2 = keyPoints2[eslesenler.trainIdx]
    # noktaları alıyorum burada
    wi_1 = int(kp1.pt[0])
    hei_1 = int(kp1.pt[1])
    wi_2 = int(kp2.pt[0])
    hei_2 = int(kp2.pt[1])

    source_points.append([hei_1, wi_1])
    destination_points.append([hei_2, wi_2])
    cv2.circle(overlay, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)

```

```

cv2.circle(overlay, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
cv2.line(overlay, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
resimImageBinaryRansacsiz[hei_1, wi_1] = [255, 255, 255]
resimImageBinaryRansacsiz[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)
resimIndex = p.split('/')[-1]
cv2.imwrite('{0}\Ransacsiz_{1}'.format(directoryName, resimIndex), img)
cv2.imwrite('{0}\Ransacsiz_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansacsiz)

# estimate affine transform model using all coordinates
model = linear_model.RANSACRegressor(linear_model.LinearRegression())
source_points = np.array(source_points)
destination_points = np.array(destination_points)
model.fit(source_points, destination_points)

# robustly estimate affine transform model with RANSAC
inliers = model.inlier_mask_
outliers = inliers == False
inlier_idxs = np.nonzero(inliers)[0]
ransacFeatureListim = []

for i in inlier_idxs:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = matches[i][1]
    kp1 = keyPoints[eslesenler.queryIdx]
    kp2 = keyPoints2[eslesenler.trainIdx]
    # noktaları alıyorum burada
    wi_1 = int(kp1.pt[0])
    hei_1 = int(kp1.pt[1])
    wi_2 = int(kp2.pt[0])
    hei_2 = int(kp2.pt[1])
    cv2.circle(ransacImage, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
    cv2.circle(ransacImage, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
    cv2.line(ransacImage, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansaccli[hei_1, wi_1] = [255, 255, 255]
    resimImageBinaryRansaccli[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(ransacImage, alpha, image2, 1 - alpha, 0, image2)
cv2.imwrite('{0}\Ransaccli_{1}'.format(directoryName, resimIndex), image2)
cv2.imwrite('{0}\Ransaccli_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansaccli)

except BaseException as e:
    print(" hata - {}", str(e))
    pass

```

#### # DAISY (SIFT\_Sift\_KNN.py) #

```

import cv2
import numpy as np
from sklearn import linear_model
import os
import random
import datetime
import Validation_Module.Validation as vld
import FeatureMatchers.RANSAC as manRANSAC

```

```
def sift_sift_knn(image_file_paths, patchSize, directoryName, thresholdForMinNeighbour=None,
thresholdForMinDistance=None):
```

```
    if not os.path.exists(directoryName):
```

```
        os.mkdir(directoryName)
```

```
    for p in image_file_paths:
```

```
        try:
```

```
            path = p
```

```
            img = cv2.imread(path)
```

```
            image2 = cv2.imread(path)
```

```
            height, width, channel = img.shape
```

```
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
            gray_logPolar100 = cv2.logPolar(gray, (gray.shape[0] / 2, gray.shape[1] / 2), 100,
```

```
                cv2.WARP_FILL_OUTLIERS)
```

```
            # gray_logPolar90 = cv2.logPolar(gray, (gray.shape[0] / 2, gray.shape[1] / 2), 90,
```

```
            #                 cv2.WARP_FILL_OUTLIERS)
```

```
            # gray_logPolar75 = cv2.logPolar(gray, (gray.shape[0] / 2, gray.shape[1] / 2), 75,
```

```
            #                 cv2.WARP_FILL_OUTLIERS)
```

```
            #
```

```
            # gray_logPolar50 = cv2.logPolar(gray, (gray.shape[0] / 2, gray.shape[1] / 2), 50,
```

```
            #                 cv2.WARP_FILL_OUTLIERS)
```

```
            # cv2.imshow("gray_logPolar100", gray_logPolar100)
```

```
            # cv2.imshow("gray_logPolar90", gray_logPolar90)
```

```
            # cv2.imshow("gray_logPolar75", gray_logPolar75)
```

```
            # cv2.imshow("gray_logPolar50", gray_logPolar50)
```

```
            # cv2.waitKey()
```

```
            cv2.imwrite("Temp\\gray_logPolar100.png", gray_logPolar100)
```

```
            resimImageBinaryRansacsiz = np.zeros((height, width, 3), np.uint8)
```

```
            resimImageBinaryRansac1 = np.zeros((height, width, 3), np.uint8)
```

```
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
            sift = cv2.xfeatures2d.SIFT_create()
```

```
            kp = sift.detect(gray, None)
```

```
            imgdraw = img.copy()
```

```
            graydraw = gray.copy()
```

```
            cv2.drawKeypoints(graydraw, kp, imgdraw)
```

```
            cv2.imwrite("{}SIFT_{}".format(directoryName, path.split('/')[-1]), imgdraw)
```

```
            siftDescriptor = cv2.xfeatures2d.SIFT_create()
```

```
            keyPoints, descriptors = siftDescriptor.compute(gray, kp)
```

```
            keyPoints2 = keyPoints.copy()
```

```
            descriptors2 = descriptors.copy()
```

```
            bf = cv2.BFMatcher()
```

```
            matches = bf.knnMatch(descriptors, descriptors2, k=3)
```

```
            good = []
```

```
            silinecek_indexList = []
```

```
            validation_total_count = len(matches)
```

```
            for index in range(0, len(matches)):
```

```
                eslesenler = matches[index][1]
```

```
                if thresholdForMinDistance is not None:
```

```
                    kp1 = keyPoints[eslesenler.queryIdx]
```

```
                    kp2 = keyPoints2[eslesenler.trainIdx]
```

```
                    # noktaları alıyorum burada
```

```
                    wi_1 = int(kp1.pt[0])
```

```
                    hei_1 = int(kp1.pt[1])
```

```
                    wi_2 = int(kp2.pt[0])
```

```
                    hei_2 = int(kp2.pt[1])
```

```
                    uzaklik = ((wi_1 - wi_2) ** 2 + (hei_1 - hei_2) ** 2) ** 0.5
```

```
                    if uzaklik <= thresholdForMinDistance:
```

```
                        silinecek_indexList.append(index)
```

```
                    continue
```

```

if thresholdForMinNeighbour is not None:
    kendisi, m, n = matches[index]
    if m.distance >= thresholdForMinNeighbour * n.distance:
        silinecek_indexList.append(index)
        continue

if len(silinecek_indexList) > 0:
    silinecek_indexList = list(
        set(silinecek_indexList))
    for index in range(0, len(matches)):
        if index not in silinecek_indexList:
            good.append(matches[index])
        else:
            silinecek = matches[index][1]
            kp1 = keyPoints[silinecek.queryIdx]
            kp2 = keyPoints2[silinecek.trainIdx]
            # noktaları alıyorum burada

if len(good) > 0:
    matches = []
    matches = good

if type(img) is not np.array:
    img = np.array(img)
    overlay = img.copy()
    ransacImage = img.copy()

alpha = 0.7

source_points = []
destination_points = []

for match in matches:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = match[1]
    kp1 = keyPoints[eslesenler.queryIdx]
    kp2 = keyPoints2[eslesenler.trainIdx]
    # noktaları alıyorum burada
    wi_1 = int(kp1.pt[0])
    hei_1 = int(kp1.pt[1])
    wi_2 = int(kp2.pt[0])
    hei_2 = int(kp2.pt[1])

    source_points.append([hei_1, wi_1])
    destination_points.append([hei_2, wi_2])
    cv2.circle(overlay, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
    cv2.circle(overlay, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
    cv2.line(overlay, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansacsiz[hei_1, wi_1] = [255, 255, 255]
    resimImageBinaryRansacsiz[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)
resimIndex = p.split('/')[ -1]
cv2.imwrite('{0}\Ransacsiz_{1}'.format(directoryName, resimIndex), img)
cv2.imwrite('{0}\Ransacsiz_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansacsiz)

model = linear_model.RANSACRegressor(linear_model.LinearRegression())
source_points = np.array(source_points)
destination_points = np.array(destination_points)
model.fit(source_points, destination_points)

inliers = model.inlier_mask_
outliers = inliers == False

```

```
inlier_idx = np.nonzero(inliers)[0]
ransacFeatureListim = []
```

```
for i in inlier_idx:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = matches[i][1]
    kp1 = keyPoints[eslesenler.queryIdx]
    kp2 = keyPoints2[eslesenler.trainIdx]
    # noktaları alıyorum burada
    wi_1 = int(kp1.pt[0])
    hei_1 = int(kp1.pt[1])
    wi_2 = int(kp2.pt[0])
    hei_2 = int(kp2.pt[1])
    cv2.circle(ransacImage, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
    cv2.circle(ransacImage, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
    cv2.line(ransacImage, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansaccli[hei_1, wi_1] = [255, 255, 255]
    resimImageBinaryRansaccli[hei_2, wi_2] = [255, 255, 255]
```

```
cv2.addWeighted(ransacImage, alpha, image2, 1 - alpha, 0, image2)
cv2.imwrite('{0}\Ransaccli_{1}'.format(directoryName, resimIndex), image2)
cv2.imwrite('{0}\Ransaccli_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansaccli)
```

```
except BaseException as e:
    print(" hata - {} ", str(e))
    pass # endregion
```

## # DAISY (SIFT\_Surf\_KNN.py) #

```
import cv2
import numpy as np
from sklearn import linear_model
import os
import random
import datetime
import Validation_Module.Validation as vld
```

```
def sift_surf_knn(image_file_paths, patchSize, directoryName, thresholdForMinNeighbour=None,
thresholdForMinDistance=None):
```

```
    if not os.path.exists(directoryName):
        os.mkdir(directoryName)
    for p in image_file_paths:
        try:
```

```
            path = p
            img = cv2.imread(path)
            image2 = cv2.imread(path)
            height, width, channel = img.shape
            resimImageBinaryRansacsiz = np.zeros((height, width, 3), np.uint8)
            resimImageBinaryRansaccli = np.zeros((height, width, 3), np.uint8)
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            sift = cv2.xfeatures2d.SIFT_create()
            kp = sift.detect(gray, None)
            imgdraw = img.copy()
            graydraw = gray.copy()
            cv2.drawKeypoints(graydraw, kp, imgdraw)
            cv2.imwrite('{0}\SIFT_{1}'.format(directoryName, path.split('/')[-1]), imgdraw)
```

```

surfDescriptor = cv2.xfeatures2d.SURF_create()

keyPoints, descriptors = surfDescriptor.compute(gray, kp)
keyPoints2 = keyPoints.copy()
descriptors2 = descriptors.copy()

bf = cv2.BFMatcher()
matches = bf.knnMatch(descriptors, descriptors2, k=3) # en yakin 2 nokta, ilki hep kendisi oluyor
zaten.

good = []
silinecek_indexList = []
validation_total_count = len(matches)
for index in range(0, len(matches)):
    eslesenler = matches[index][1]
    if thresholdForMinDistance is not None:
        kp1 = keyPoints[eslesenler.queryIdx]
        kp2 = keyPoints2[eslesenler.trainIdx]
        # noktaları alıyorum burada
        wi_1 = int(kp1.pt[0])
        hei_1 = int(kp1.pt[1])
        wi_2 = int(kp2.pt[0])
        hei_2 = int(kp2.pt[1])
        uzaklik = ((wi_1-wi_2)**2+(hei_1-hei_2)**2)**0.5
        if uzaklik<= thresholdForMinDistance:
            silinecek_indexList.append(index)
            continue
    if thresholdForMinNeighbour is not None:
        kendisi,m, n = matches[index]
        if m.distance >= thresholdForMinNeighbour * n.distance:
            silinecek_indexList.append(index)
            continue
    if len(silinecek_indexList)>0:
        silinecek_indexList = list(set(silinecek_indexList)) # listeyi unique hale getiriyorumki aynı değerler
        elensin.
    for index in range(0, len(matches)):
        if index not in silinecek_indexList:
            good.append(matches[index])
        else:
            silinecek = matches[index][1]
            kp1 = keyPoints[silinecek.queryIdx]
            kp2 = keyPoints2[silinecek.trainIdx]
            # noktaları alıyorum burada
            wi_s1 = int(kp1.pt[0])
            hei_s1 = int(kp1.pt[1])
            wi_s2 = int(kp2.pt[0])
            hei_s2 = int(kp2.pt[1])

    if len(good)>0:
        matches = []
        matches = good

    if type(img) is not np.array:
        img = np.array(img)
    overlay = img.copy()
    ransacImage = img.copy()
    alpha = 0.7
    source_points = []
    destination_points = []
    for match in matches:
        color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
        eslesenler = match[1]
        kp1 = keyPoints[eslesenler.queryIdx]
        kp2 = keyPoints2[eslesenler.trainIdx]
        # noktaları alıyorum burada

```

```

wi_1 = int(kp1.pt[0])
hei_1 = int(kp1.pt[1])
wi_2 = int(kp2.pt[0])
hei_2 = int(kp2.pt[1])
source_points.append([hei_1, wi_1])
destination_points.append([hei_2, wi_2])
cv2.circle(overlay, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
cv2.circle(overlay, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
cv2.line(overlay, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
resimImageBinaryRansacsiz[hei_1, wi_1] = [255, 255, 255]
resimImageBinaryRansacsiz[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)
resimIndex = p.split('/')[-1]
cv2.imwrite('{0}\Ransacsiz_{1}'.format(directoryName, resimIndex), img)
cv2.imwrite('{0}\Ransacsiz_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansacsiz)

model = linear_model.RANSACRegressor(linear_model.LinearRegression())
source_points = np.array(source_points)
destination_points = np.array(destination_points)
model.fit(source_points, destination_points)

inliers = model.inlier_mask_
outliers = inliers == False
inlier_idx = np.nonzero(inliers)[0]
ransacFeatureListim = []

for i in inlier_idx:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = matches[i][1]
    kp1 = keyPoints[eslesenler.queryIdx]
    kp2 = keyPoints2[eslesenler.trainIdx]
    # noktaları alıyorum burada
    wi_1 = int(kp1.pt[0])
    hei_1 = int(kp1.pt[1])
    wi_2 = int(kp2.pt[0])
    hei_2 = int(kp2.pt[1])
    cv2.circle(ransacImage, (wi_1, hei_1), 5, (color[0], color[1], color[2]), 0)
    cv2.circle(ransacImage, (wi_2, hei_2), 5, (color[0], color[1], color[2]), 0)
    cv2.line(ransacImage, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansaccli[hei_1, wi_1] = [255, 255, 255]
    resimImageBinaryRansaccli[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(ransacImage, alpha, image2, 1 - alpha, 0, image2)
cv2.imwrite('{0}\Ransaccli_{1}'.format(directoryName, resimIndex), image2)
cv2.imwrite('{0}\Ransaccli_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansaccli)

except BaseException as e:
    print(" hata - {}".format(e))
    pass # endregion

```

#### # TEZ ÇALIŞMASI Python Modülü (SIFT\_Fatih\_LBP\_KNN.py) #

```

import os
from datetime import datetime

import numpy as np
from sklearn import linear_model
import cv2
from CornerDetection import CornerDetectionModule as cdm
from Classes.cs_Feature import *

```

```

# from skimage import feature
import CModules.VBSD_C_Module.myvbsd as myvbsd
import CModules.LBP_C_Module.mylbp as mylbp
import random
from skimage import feature
import FeatureMatchers.RANSAC as manRANSAC

is_roll_enabled = False
calculateLogPolarCoor = False

# 256 / 3 yi gecmesin histogramların toplam size
# default olarak 8,16,24 kullan 48*3 = 144 gidiyor sonuc
sekizBirHistogramSize = 8 # 8
onAltıKiHistogramSize = 16 # 16
yimirDortUcHistogramSize = 24 # 24
CompareMethods = {
    "Euclid": 0,
    "NCC": 1,
    "ChiSq": 2,
    "CosineDist": 3,
    "Manhattan": 4,
    "JeffreyDivergence": 5
}
DEBUG = {
    "OFF": 0,
    "ON": 1,
}
debug = DEBUG["OFF"]

def sift_lbp(image_file_paths, patchSize, directoryName, thresholdForMinNeighbour,
thresholdForMinDistance,
ThresholdForGLCM=None, histCount=None, compMethodFromOut = None):
    if not os.path.exists(directoryName):
        os.mkdir(directoryName)
    for p in image_file_paths:
        try:
            # 16 -> 8, 32 -> 12, 80 -> 16
            # path = relevant_path+'/' + p
            print("*****")
            print("SIFT Basladi {:%H:%M:%S} , {}".format(datetime.now(), str(p)))
            featureListim, mag_ori_image = cdm.Sift(p, patchSize, directoryName,
getMagnitudeAndOrientation=True,
useGaussBlurFilter=True, calculateLogPolar=calculateLogPolarCoor)
            print("SIFT Bitti {:%H:%M:%S} ".format(datetime.now()))
            koorListim = []
            patchDataList = []
            #featureListim = featureListim[0:5]

            resultHistograms = list(map(CreateLBPHistograms, featureListim))
            koorListim = np.asarray([(x[0], x[1]) for x in resultHistograms], np.int)
            patchDataList = np.asarray([x[2] for x in resultHistograms], np.float)

            # region VBSD Match
            #
            compareMethod = CompareMethods["Manhattan"]
            histogramCount = 2
            if histCount is not None:
                histogramCount = histCount
            if compMethodFromOut is not None:
                compareMethod = compMethodFromOut
            initialArrSize = len(patchDataList[0])
            out_arrayIndexes, out_arrayValues = myvbsd.arrFunc2d(koorListim, patchDataList, histogramCount,
thresholdForMinDistance,
thresholdForMinNeighbour, initialArrSize, debug,
compareMethod)

```

```

if compareMethod in [CompareMethods["Euclid"],
    CompareMethods["ChiSq"], CompareMethods["Manhattan"],
    CompareMethods["JeffreyDivergence"]]:
    out_arrayResult = [
        int(x / 2), int(out_arrayIndexes[x]), int(out_arrayIndexes[x + 1]), out_arrayValues[x],
        out_arrayValues[x + 1],
        out_arrayValues[x] / out_arrayValues[x + 1] for x in range(0, len(out_arrayIndexes), 2)
        if out_arrayValues[x] / out_arrayValues[x + 1] <= thresholdForMinNeighbour]
elif compareMethod in [CompareMethods["NCC"],
    CompareMethods["CosineDist"]]:
    # burada bunu yapmamdaki amac ncc en yakinini almasini istememdir.
    out_arrayResult = [
        int(x / 2), int(out_arrayIndexes[x]), int(out_arrayIndexes[x + 1]), out_arrayValues[x],
        out_arrayValues[x + 1],
        out_arrayValues[x] for x in range(0, len(out_arrayIndexes), 2)
        if out_arrayValues[x] >= thresholdForMinNeighbour]

# endregion

img = cv2.imread(p)
img2 = img.copy()
overlay = img.copy()
ransacImage = img.copy()

height, width, channel = img.shape
resimImageBinaryRansacsiz = np.zeros((height, width, 3), np.uint8)
resimImageBinaryRansaccli = np.zeros((height, width, 3), np.uint8)

alpha = 1
# region RansacIslemleri
source_points = []
destination_points = []
for match in out_arrayResult:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = match
    kp1 = koorListim[eslesenler[0]]
    kp2 = koorListim[eslesenler[1]]
    # noktaları alıyorum burada
    wi_1 = int(kp1[0])
    hei_1 = int(kp1[1])
    wi_2 = int(kp2[0])
    hei_2 = int(kp2[1])
    source_points.append([hei_1, wi_1])
    destination_points.append([hei_2, wi_2])
    cv2.circle(overlay, (wi_1, hei_1), 4, (color[0], color[1], color[2]), -1)
    cv2.circle(overlay, (wi_2, hei_2), 4, (color[0], color[1], color[2]), -1)
    cv2.line(overlay, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansacsiz[hei_1, wi_1] = [255, 255, 255]
    resimImageBinaryRansacsiz[hei_2, wi_2] = [255, 255, 255]

cv2.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)
# cv2.imshow("overlay", overlay)
# cv2.imshow("img", img)
# cv2.waitKey(0)
resimIndex = p.split("/")[-1]
cv2.imwrite('{0}\'Ransacsiz_{1}\''.format(directoryName, resimIndex), img)
cv2.imwrite('{0}\'Ransacsiz_BImage_{1}\''.format(directoryName, resimIndex),
resimImageBinaryRansacsiz)

# estimate affine transform model using all coordinates
model = linear_model.RANSACRegressor(linear_model.LinearRegression())
source_points = np.array(source_points)
destination_points = np.array(destination_points)
model.fit(source_points, destination_points)

```

```

# robustly estimate affine transform model with RANSAC
inliers = model.inlier_mask_
outliers = inliers == False
inlier_idx = np.nonzero(inliers)[0]
ransacFeatureListim = []

for i in inlier_idx:
    color = [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)]
    eslesenler = out_arrayResult[i]
    kp1 = koorListim[eslesenler[0]]
    kp2 = koorListim[eslesenler[1]]
    # noktaları alıyorum burada
    wi_1 = int(kp1[0])
    hei_1 = int(kp1[1])
    wi_2 = int(kp2[0])
    hei_2 = int(kp2[1])

    cv2.circle(ransacImage, (wi_1, hei_1), 4, (color[0], color[1], color[2]), -1)
    cv2.circle(ransacImage, (wi_2, hei_2), 4, (color[0], color[1], color[2]), -1)
    cv2.line(ransacImage, (wi_1, hei_1), (wi_2, hei_2), (color[0], color[1], color[2]))
    resimImageBinaryRansac[i][wi_1] = [255, 255, 255]
    resimImageBinaryRansac[i][hei_2, wi_2] = [255, 255, 255]

    cv2.addWeighted(ransacImage, alpha, img2, 1 - alpha, 0, img2)
    cv2.imwrite('{0}\Ransac_{1}'.format(directoryName, resimIndex), img2)
    cv2.imwrite('{0}\Ransac_BImage_{1}'.format(directoryName, resimIndex),
resimImageBinaryRansac)

    print("*****- {0}".format(len(out_arrayResult)))
    print(out_arrayResult)
    print("*****")

except BaseException as e:
    print("Hata {}".format(str(e)))
    pass

```

#### def CreateLBPHistograms(feature):

```

patchDataMagnitudeSekizBir, patchDataMagnitudeSekizBirHistogram, patchDataMagnitudeOnAltıİki \
    , patchDataMagnitudeOnAltıİkiHistogram, patchDataMagnitudeYirmiDörtÜç,
patchDataMagnitudeYirmiDörtÜçHistogram = getLBPHistogram(
    feature.patchDataMagnitude)

patchDataOrientationSekizBir, patchDataOrientationSekizBirHistogram, patchOrientationOnAltıİki \
    , patchDataOrientationOnAltıİkiHistogram, patchDataOrientationYirmiDörtÜç,
patchDataOrientationYirmiDörtÜçHistogram = getLBPHistogram(
    feature.patchDataOrientation, lbpMethod="Default")

patchDataSekizBir, patchDataSekizBirHistogram, patchDataOnAltıİki \
    , patchDataOnAltıİkiHistogram, patchDataYirmiDörtÜç, patchDataYirmiDörtÜçHistogram =
getLBPHistogram(
    feature.patchData)

npPatchDatas = np.concatenate((np.ravel(patchDataSekizBirHistogram[0]),
np.ravel(patchDataOnAltıİkiHistogram[0]),
np.ravel(patchDataYirmiDörtÜçHistogram[0])), axis=None)

npPatchDatasMagnitude = np.concatenate(
    (np.ravel(patchDataMagnitudeSekizBirHistogram[0]),
np.ravel(patchDataMagnitudeOnAltıİkiHistogram[0]),
np.ravel(patchDataMagnitudeYirmiDörtÜçHistogram[0])), axis=None)

npPatchDatasOrientation = np.concatenate(

```

```

(np.ravel(patchDataOrientationSekizBirHistogram[0]),
np.ravel(patchDataOrientationOnAltiIkiHistogram[0]),
np.ravel(patchDataOrientationYirmiDortUcHistogram[0])), axis=None)

npPatchDatasResult = np.concatenate((npPatchDatas, npPatchDatasMagnitute, npPatchDatasOrientation))
# npPatchDatasResult = np.ravel(patchDataYirmiDortUcHistogram[0]) # Orjinal 16,2
return feature.pointX, feature.pointY, npPatchDatasResult

```

```

def createLBPVal(patchData, h, w):
    center = patchData[h, w]
    val = 0
    index = 0
    newArr = [0] * 8
    for ih in range(h - 1, h + 2):
        for iw in range(w - 1, w + 2):
            if ih == h and iw == w: # kendisini eylelim
                continue
            data = 0
            if (ih >= 0 and iw >= 0) and (ih < patchData.shape[0] and iw < patchData.shape[1]):
                data = patchData[ih, iw]
                newArr[index] = data
            index += 1
    for i in range(0, 8):
        val1 = newArr[i % 8]
        val2 = newArr[(i + 1) % 8]
        if val1 > val2:
            val += 1 << i
    return val

```

```

def getLBPHistogram(data1, lbpMethod="Default"):
    pd_data1 = np.asarray(data1, np.int)

    npzeros1 = np.zeros([pd_data1.shape[0] + 2, pd_data1.shape[1] + 2], dtype=np.int)
    npzeros1[1:-1, 1:-1] = pd_data1
    npzeros2 = np.zeros([pd_data1.shape[0] + 4, pd_data1.shape[1] + 4], dtype=np.int)
    npzeros2[2:-2, 2:-2] = pd_data1
    npzeros3 = np.zeros([pd_data1.shape[0] + 6, pd_data1.shape[1] + 6], dtype=np.int)
    npzeros3[3:-3, 3:-3] = pd_data1

    # mode = 0 => normal LBP
    # mode = 1 => ror Min
    # mode = 2 => ror Max
    mode = 0
    if lbpMethod == "rorMin":
        mode = 1
    elif lbpMethod == "rorMax":
        mode = 2
    pd_sekiz_bir = mylbp.CalcLBP(npzeros1, 1, 8, mode, debug)[1:-1,
        1:-1] # ilk tarafta 0 lar cevriili oldugu icin ekliyorum
    pd_onalti_iki = mylbp.CalcLBP(npzeros2, 2, 16, mode, debug)[2:-2,
        2:-2] # ilk tarafta 0 lar cevriili oldugu icin ekliyorum
    pd_yirmidort_uc = mylbp.CalcLBP(npzeros3, 3, 24, mode, debug)[3:-3,
        3:-3] # ilk tarafta 0 lar cevriili oldugu icin ekliyorum

    h_pd_sekiz_bir = np.histogram(pd_sekiz_bir, bins=sekizBirHistogramSize, range=(0, 256))
    h_pd_onalti_iki = np.histogram(pd_onalti_iki, bins=onAltiIkiHistogramSize,
        range=(0, 65536))
    h_pd_yirmidort_uc = np.histogram(pd_yirmidort_uc, bins=yimirDortUcHistogramSize,
        range=(0, 16777216))
    return pd_sekiz_bir, h_pd_sekiz_bir, pd_onalti_iki, h_pd_onalti_iki, pd_yirmidort_uc, h_pd_yirmidort_uc

```

**# TEZ ÇALIŞMASI LBP C Kodu (LBPModule.c) #**

```

#include <stdio.h>
#include <math.h>
#include "LBPModule.h"
#include <stdint.h>

#define MIN(a,b) ( (a<b) ? a : b )
#define MAX(a,b) ( (a>b) ? a : b )
#define M_PI 3.14159265358979323846

typedef struct
{
    unsigned int  int8 : 8;
    unsigned int  int16 : 16;
    unsigned int  int24 : 24;
    unsigned int  int32;
}lbpShiftSt;

//arr[i][j] = *(ptr + (i x no_of_cols + j))

typedef enum {org = 0, rorMin = 1, rorMax = 2 }lbpMode;

int GetShiftedValue(int val,int neighbour,int shiftCount){
    // Circular shift yapıyorum
    lbpShiftSt lbpShift;
    switch(neighbour){
        case 8:
            lbpShift.int8 = (val >> shiftCount) | (val << ( 8 - shiftCount) );
            return lbpShift.int8;
        case 16:
            lbpShift.int16 = (val >> shiftCount) | (val << ( 16 - shiftCount) );
            return lbpShift.int16 ;
        case 24:
            lbpShift.int24 = (val >> shiftCount) | (val << ( 24 - shiftCount) );
            return lbpShift.int24 ;
        case 32:
            lbpShift.int32 = (val >> shiftCount) | (val << ( 32 - shiftCount) );
            return lbpShift.int32;
        default:
            return 0;
    }
}

/*
    mode = 0 => normal LBP
    mode = 1 => ror Min
    mode = 2 => ror Max
    mode = 3 => cslbp ?
*/

void applyLocalBinaryPattern(int* source, int *destination, int radius,int neighbours
,int rowNumber, int colNumber,int mode, int debug){

    neighbours = MAX(MIN(neighbours,31),1);
    int n ;
    float maxVal = 0 ;
    float minVal = pow(neighbours,2) + 1 ;
    int maxIndex, minIndex ;

    int i,j ;

```

```

    for(i = radius; i < rowNumber - radius;i++) {
    for(j = radius; j < colNumber - radius;j++) {
    int lbpVal = 0;
    int centerVal = *(source + (i * colNumber) + j);
    if(debug){
        printf("Center Point (R,C) = (%d, %d) = %d \n",i,j,centerVal);
    }

    minIndex = 0 ;
    maxIndex = 0 ;
    for(n = 0 ; n < neighbours ; n++){
        // sample points
        float x = (float)radius * cos(2.0 * M_PI * n / (float)neighbours) ;
        float y = (float)radius * -sin(2.0 * M_PI * n / (float)neighbours) ;
        // relative indices
        int fx = (int)floor(x);
        int fy = (int)floor(y);
        int cx = (int)ceil(x);
        int cy = (int)ceil(y);
        //fractional part
        float ty = y - fy;
        float tx = x - fx;
        //set interpolation weights
        float w1 = (1 - tx) * (1 - ty);
        float w2 = tx * (1 - ty);
        float w3 = (1 - tx) * ty;
        float w4 = tx * ty;

        if (debug){
            printf("\n");
            printf("i: %d, j:%d \n",i,j);
            printf("w1:%f - %d - %d \n",w1,i+fy,j+fx);
            printf("w2:%f - %d - %d \n",w2,i+fy,j+cx);
            printf("w3:%f - %d - %d \n",w3,i+cy,j+fx);
            printf("w4:%f - %d - %d \n",w4,i+cy,j+cx);
            printf("\n");
        }

        //int *centerPoint =
        int point1Location = (i+fy) * colNumber + j+fx ;
        int point2Location = (i+fy) * colNumber + j+cx ;
        int point3Location = (i+cy) * colNumber + j+fx ;
        int point4Location = (i+cy) * colNumber + j+cx ;

        int point1 = *(source + point1Location);
        int point2 = *(source + point2Location);
        int point3 = *(source + point3Location);
        int point4 = *(source + point4Location);

        if (debug){
            printf("Point1 : %d \n", point1);
            printf("Point2 : %d \n", point2);
            printf("Point3 : %d \n", point3);
            printf("Point4 : %d \n", point4);
        }
        float val = w1 * point1 + w2 * point2 + w3 * point3 + w4 * point4 ;

        if(val > maxVal && mode == (int)rorMax){
            maxVal = val;
            maxIndex = n;
        }
        else if(val < minVal && mode == (int)rorMin){
            minVal = val;
            minIndex = n ;
        }
    }
}

```

```

        if(val >= centerVal){
            lbpVal += (1<<n);
        }

    }

    int lbpValBS;
    switch(mode){
        case 0:
        default:
            break;
        case (int)rorMin:
            lbpValBS = lbpVal;
            lbpVal = GetShiftedValue(lbpVal, neighbours, minIndex);

            /*shiftCount = minIndex;
            //lbpVal = lbpVal >> minIndex;
            if(debug){
                //printf("\n ShiftNumber : %d",*shiftCount);
                printf("\nMinIndex Index = %d, before shift LBPPVAL = %d
, after shift LBPPVAL = %d \n",minIndex,lbpValBS,lbpVal);
            }
            break;
        case (int)rorMax:
            lbpValBS = lbpVal;
            //lbpVal = lbpVal >> maxIndex; // maxIndex basa gelicek kadar shift
            lbpVal = GetShiftedValue(lbpVal, neighbours, maxIndex);

            /*shiftCount = maxIndex;
            if(debug){
                //printf("\n ShiftNumber : %d",*shiftCount);
                printf("\nMaxIndex Index = %d, before shift LBPPVAL = %d
, after shift LBPPVAL = %d \n",maxIndex,lbpValBS,lbpVal);
            }
            break;
    }
    *(destination + (i * colNumber) + j) = lbpVal ;
    if(debug){
        printf("\n*****\n");
    }
}
}

}

```

**# TEZ ÇALIŞMASI LBP C Modulune ait Python bağlayıcı sınıfı (p\_lbp.pyx) #**

```

import numpy as np
cimport numpy as np

```

```

cdef extern from "LBPMModule.h":

```

```

    void applyLocalBinaryPattern(int* source, int *destination, int radius,int neighbours
,int rowNumber, int colNumber,int mode ,int debug)

```

```
def CalcLBP(np.ndarray[np.int_t, ndim=2] image, radius, neighbours , mode, debug):
    cdef np.ndarray out_array = np.zeros([image.shape[0], image.shape[1]], dtype = np.int)
    #cdef int* shift
    applyLocalBinaryPattern(<int*> image.data, <int*> out_array.data, radius, neighbours
        ,image.shape[0],image.shape[1],mode ,debug)
    #shiftCount = int(shift[0])
    #print(shiftCount)
    return out_array
```

#### # TEZ ÇALIŞMASI VBSD C Kodu (LBPModule.c) #

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include "vbsd.h"

typedef enum {Euclid = 0 , NCC = 1 ,
              ChiSq = 2, CosineDist = 3, Manhattan = 4, JeffreyDivergence = 5} myMatchMethod;

void CalculateVBSDDistance(int* arr1Coordinates, int* arr2Coordinates,
double* arrImageData1, double* arrImageData2,
int histogramCount, int thresholdForMinDistance, int thresholdForMinNeighbour,
int itemCount, int initialArrSize, int debug, int matchMethod ,int* outResultIndexes, double*
outResultValues ){
    int i;
    double arr1Malloc[256]; // ne kadar kullandigimdan ziyade ne kullandigim onemli
    double arr2Malloc[256];

    int resultIndex = 0 ;
    for(i = 0 ; i < itemCount ; i +=1 ){
        int xKoor1 = *(arr1Coordinates + (2 * i) );
        int yKoor1 = *(arr1Coordinates + (2 * i) + 1 );
        int j;

        int bestIndexes[2] = {0};
        double bestValues[2] = {0};
        bestIndexes[0] = -1;
        bestIndexes[1] = -1;
        bestValues[0] = INT_MAX;
        bestValues[1] = INT_MAX;
        if(matchMethod == NCC || matchMethod == CosineDist){
            bestValues[0] = INT_MIN;
            bestValues[1] = INT_MIN;
        }

        for(j = 0 ; j < itemCount ; j +=1 ){
            if(j == i){
                continue;
            }

            int arraySize = initialArrSize; // inti
            double sum = 0;
            int xKoor2 = *(arr1Coordinates + (2 * j) );
            int yKoor2 = *(arr1Coordinates + (2 * j) + 1 );
            double distanceBetweenTwoPoints =
CalculateEuclidBetweenTwoPoints(xKoor1,xKoor2,yKoor1,yKoor2);
```

```

        if(thresholdForMinDistance > 0 &&
            distanceBetweenTwoPoints < thresholdForMinDistance){
            // Eger birbirlerine yakinsalar mesafe olarak karsilastirmaya
almyyacagim
                continue;
        }
        // arrayleri burada yeni bir arraye kopyalacagim oradan islem yapicam en iyisi
        memset(arr1Malloc, 0, initialArrSize*sizeof(arr1Malloc[0]));
        memset(arr2Malloc, 0, initialArrSize*sizeof(arr2Malloc[0]));
        memcpy(&arr1Malloc[0], &arrImageData1[0]+ initialArrSize * i,
sizeof(double)*initialArrSize ); // burada memcpy yaparken hep bir sonrakini alican arrImageData1 + arraySize
* i
        memcpy(&arr2Malloc[0], &arrImageData2[0] + initialArrSize * j,
sizeof(double)*initialArrSize );// burada memcpy yaparken hep bir sonrakini alican arrImageData2 + arraySize
* j

        if(debug == 1){
            printf("\n***** Processing : %d - %d\n",i,j);
            printf("\n-----After Malloc sonra arrayler-----");
            PrintArr(&arr1Malloc[0],arraySize);
            PrintArr(&arr2Malloc[0],arraySize);
        }

        int histIndex ;
        for(histIndex = 0 ; histIndex < histogramCount ; histIndex++){
            if(histIndex > 0){
                // burada intersection islemleri yapilacaktir.
                FindAndSubtractIntersectionFromArrays(arr1Malloc,
arr2Malloc, arraySize);

                //burada initialSize = initialSize / 2 olacak ve
                // ardisik datalar toplanacak.
                SumConsecutiveDatasInArrays(arr1Malloc, arr2Malloc,
arraySize);

                arraySize = (arraySize / 2);

                if(debug == 1){
                    printf("\n-----After Merge Consecutive Cells #%-
-----",histIndex);
                    PrintArr(&arr1Malloc[0],arraySize);
                    PrintArr(&arr2Malloc[0],arraySize);
                }
            }
        }

        // arrayler arasi euclid distance bakicam simdi
        double distanceVal;
        switch(matchMethod){
            case Euclid:
                // Euclid
                distanceVal =
CalculateEuclidDistanceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
                if(debug == 1){
                    printf("\n-----Euclid Distance: %f -----",distanceVal);
                }

                break;
            case NCC:

```

```

        // NCC
        distanceVal =
CalculateNCCistanceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
        if(debug == 1){
            printf("\n-----NCC Distance: %f -----",distanceVal);
        }
        break;
    case ChiSq:
        distanceVal =
CalculateChiSquareDistanceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
        if(debug == 1){
            printf("\n-----ChiSq Distance: %f -----",distanceVal);
        }
        break;
    case CosineDist:
        distanceVal =
CalculateCosineDistanceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
        if(debug == 1){
            printf("\n-----CosineDist Distance: %f -----",distanceVal);
        }
        break;
    case Manhattan:
        distanceVal =
CalculateManhattanDistanceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
        if(debug == 1){
            printf("\n-----Manhattan Distance: %f -----",distanceVal);
        }
        break;
    case JeffreyDivergence:
        distanceVal =
CalculateJeffreyDivergenceBetweenTwoArrays(&arr1Malloc[0], &arr2Malloc[0], arraySize);
        if(debug == 1){
            printf("\n-----JeffreyDivergence Distance: %f -----",distanceVal);
        }
        break;

    default:
        break;

}
sum += distanceVal ;

}

sum = sum / histogramCount;
CheckNeighbourhood(&bestIndexes[0],&bestValues[0], j, sum, matchMethod);

// burada euclid sumlari / histogram counta bolup bulucan tam yakinlik derecesini.
if(debug == true){
    printf("\nAVG : %d - %d - %f \n",i,j,sum);
    printf("\n-----BEST INDEXES FOR %d-----", i);
    int bi;
    for(bi = 0; bi < 2 ; bi++){
        printf("\n-----Index:   %d   -   Best   Value:   %f   -----",bestIndexes[bi],bestValues[bi]);
    }
    printf("\n");
}

}

} //end for j

outResultIndexes[resultIndex] = bestIndexes[0];
outResultValues[resultIndex] = bestValues[0];
outResultIndexes[resultIndex+1] = bestIndexes[1];

```

```

        outResultValues[resultIndex+1] = bestValues[1];
        resultIndex += 2 ;
    }
}

```

```

void CheckNeighbourhood(int* bestIndexes, double* bestValues, int currIndex, double curValue, int compMethod){

```

```

    switch(compMethod){
        case Euclid:
        case ChiSq:
        case Manhattan:
        case JeffreyDivergence:
            //Euclid, chiSq
            if( curValue < bestValues[0]){

                bestValues[1] = bestValues[0];
                bestIndexes[1] = bestIndexes[0];
                bestValues[0] = curValue;
                bestIndexes[0] = currIndex;

            }
            else if( curValue < bestValues[1]){
                bestValues[1] = curValue;
                bestIndexes[1] = currIndex;
            }

            break;
        case NCC:
        case CosineDist:
            // NCC -> burada buyuk olan deger makbul
            if( curValue > bestValues[0]){

                bestValues[1] = bestValues[0];
                bestIndexes[1] = bestIndexes[0];
                bestValues[0] = curValue;
                bestIndexes[0] = currIndex;

            }
            else if( curValue > bestValues[1]){
                bestValues[1] = curValue;
                bestIndexes[1] = currIndex;
            }

            break;
    }
}

```

```

void PrintArr(double* arr, int size){
    int index = 0;
    printf("\n");
    for(index = 0 ; index < size ; index++ ){
        //printf("%f\n",arr[index]);
        printf("%f, ",*(&arr[0]+index));
    }
}

```

```

void SumConsecutiveDatasInArrays(double* arrImageData1, double* arrImageData2, int size){
    int index ;
    for(index = 0 ; index < size / 2 ; index+=1 ){
        *(arrImageData1 + index) = *(arrImageData1 + (2 * index)) + *(arrImageData1 + (2 * index) +1);
    }
}

```

```

        *(arrImageData2 + index) = *(arrImageData2 + (2 * index)) + *(arrImageData2 + (2 * index) + 1);
    }

}

```

```

double CalculateEuclidDistanceBetweenTwoArrays(double* data1, double* data2, int size){
    int index ;
    double sum = 0;
    for(index = 0 ; index < size ; index++){

        double number1 = *(data1 + index);
        double number2 = *(data2 + index);
        sum += pow(number1 -number2,2) ;
    }
    return sqrt (sum ) ;
}

```

```

double CalculateNCCistanceBetweenTwoArrays(double* data1, double* data2, int size){

    int index ;
    double sumOfMultiplies = 0;
    double sumOfSquareFirst = 0;
    double sumOfSquareSecond = 0;
    for(index = 0 ; index < size ; index++){

        double number1 = *(data1 + index);
        double number2 = *(data2 + index);
        sumOfMultiplies += (number1 * number2);
        sumOfSquareFirst += pow(number1,2);
        sumOfSquareSecond += pow(number2,2);
    }
    double sqrtOf_Multiplication_sumOfSquareFirst_sumOfSquareSecond = sqrt(sumOfSquareFirst *
sumOfSquareSecond);
    double result = 0;
    if(sqrtOf_Multiplication_sumOfSquareFirst_sumOfSquareSecond != 0)
    {
        result = (sumOfMultiplies / sqrtOf_Multiplication_sumOfSquareFirst_sumOfSquareSecond);
    }
    return result;
}

```

// VBSD makalesinden gelen formül uzerine

```

double CalculateChiSquareDistanceBetweenTwoArrays(double* data1, double* data2, int size){
    int index ;
    double result = 0;
    for(index = 0 ; index < size ; index++){

        double number1 = *(data1 + index);
        double number2 = *(data2 + index);
        double diffSquare = pow((number1-number2) , 2);
        double sum = 2 * ( number1 + number2 ) ;
        double lastRes = 0;
        if(diffSquare != 0 && sum!=0 ){
            lastRes = (diffSquare / sum);
        }
        result += lastRes;
    }
    return result;
}

```

// VBSD makalesinden gelen formül uzerine

```

double CalculateManhattanDistanceBetweenTwoArrays(double* data1, double* data2, int size){
    int index ;
    double result = 0;

```

```

for(index = 0 ; index < size ; index++ ){

    double number1 = *(data1 + index);
    double number2 = *(data2 + index);
    result += abs (number1 - number2);
}
return result;
}

// VBSD makalesinden gelen formül uzerine
double CalculateJeffreyDivergenceBetweenTwoArrays(double* data1, double* data2, int size){
    int index ;
    double result = 0;
    for(index = 0 ; index < size ; index++ ){

        double number1 = *(data1 + index);
        double number2 = *(data2 + index);
        if(number1 == 0){
            number1 = 1;
        }
        if(number2 == 0){
            number2 = 1;
        }
        double first = number1 * (log(number1) - log(number2));
        double second = number2 * (log(number2) - log(number1));
        result += (first + second);
    }
    return result;
}

double CalculateCosineDistanceBetweenTwoArrays(double* data1, double* data2, int size){
    int index ;
    double result = 0;
    double sumOfMultipleOf12 ;
    double squareOfSumOf1 ;
    double squareOfSumOf2 ;
    for(index = 0 ; index < size ; index++ ){

        double number1 = *(data1 + index);
        double number2 = *(data2 + index);
        sumOfMultipleOf12 += (number1 * number2);
        squareOfSumOf1 += pow(number1,2);
        squareOfSumOf2 += pow(number2,2);

    }
    squareOfSumOf1 = sqrt(squareOfSumOf1);
    squareOfSumOf2 = sqrt(squareOfSumOf2);
    double div = squareOfSumOf1 * squareOfSumOf2;
    if(div == 0){
        div = 1 ;
    }
    double subResult = sumOfMultipleOf12 / div;
    return 1 - subResult ;
}

double FindMinimumNumber(double n1, double n2){
    if(n1 < n2){
        return n1;
    }
    return n2;
}

```

```

void FindAndSubtractIntersectionFromArrays(double* arrImageData1, double* arrImageData2, int size){
    int index ;
    for(index = 0 ; index < size ; index++ ){
        double intersection = FindMinimumNumber(*(arrImageData1 + index), *(arrImageData2
+ index));
        *(arrImageData1 + index) = *(arrImageData1 + index) - intersection;
        *(arrImageData2 + index) = *(arrImageData2 + index) - intersection;
    }
}

```

```

double CalculateEuclidBetweenTwoPoints(int x1, int x2, int y1, int y2){

```

```

    double diffx = x1 - x2;
    double diffy = y1 - y2;
    double diffx_sqr = pow(diffx,2);
    double diffy_sqr = pow(diffy,2);
    double distance = sqrt (diffx_sqr + diffy_sqr);
    return distance;
}

```